A Phoneme Based Speech Recognition System for High Stress Moderate Noise Environments

Phase II Final Report
November 1, 1990

M.T. Anikst, B.M. Davis, W.S. Meisel, L.S. Olorenshaw, S.S. Pirzadeh, J.E. Schumacher, M.C. Soares, D.J. Trawick

Sponsored by
National Aeronautics and Space Administration,
Johnson Space Center
Small Business Innovation Research Program
Under Contract Number NAS 9-17994

SSI Document # 10046

Name of Contractor: Speech Systems Incorporated
Business Address: 18356 Oxnard Street
Tarzana, CA 91356

Phone: (818) 881-0885

Principal Investigator: David J. Trawick, Ph.D.

Reporting Period: May 23, 1988 through November 1, 1990

PROJECT SUMMARY

The main goal of this project was to develop for NASA a high performance phoneme based speech recognition development testbed, and by so doing improve SSI's core technology. The main areas of research were: (1) improvements for NASA specific requirements, and (2) general performance improvements to SSI's speech recognizer. NASA's requirements included primarily performance improvements in stressful and noisy speaking environments. These requirements are important to NASA's potential applications, but are also important to applications other than NASA's. The general performance research included studies to improve the speed, accuracy, and speaker coverage and/or independence of SSI's speech recognizer.

Several areas of research have been investigated as part of this project. A speaker model adaptation method, called profiling, has been studied. This method makes it possible to develop generic speech models with data from many speakers, and then to customize speech models for a particular speech environment such as a new speaker, group of speakers, application, dialect, or level of background noise. Profiling was found to be successful in adapting a generic model for General American English to speakers of the Southern dialect. It was also successfully used to recover most of the performance lost in noisy environment tests. A part of the profiling algorithm was used to adapt to a new headset input device. Performance of the generic models, the starting point for customization, has also been improved.

Research was performed in several areas of acoustic processing, to further increase the system's noise robustness. A constant framing rate acoustic processor was found to be superior to the former pitch synchronous framing, and better suited to model adaptation. Smoothing methods on acoustic parameters improved the performance of other stages of processing following the acoustic processor. An alternate scheme of acoustic parameterization, using cepstrum parameterization instead of bandpass filterbank parameterization, was found to yield essentially equivalent performance.

Parallel processing was investigated as a means of decreasing the speech decoding time. SSI's Phonetic Decoder was reengineered in preparation for porting to a parallel processor. The result of this effort was to decrease decoding time by about 75-85%. An expression was derived for estimating the decoding time change from porting to a parallel processing environment.

A system incorporating the results of this research has been consolidated, delivered and installed at NASA Johnson Space Center, with performance improvements similar to those predicted.

Potential commercial applications of this research include any real speech recognition applications requiring robust, accurate, and fast recognition in stressful, noisy, or most any other speech environment.

TABLE OF CONTENTS

1. INTRODUCTORY PROJECT OVERVIEW

1.1. Introduction

The main goal of this project was to develop for NASA a high performance phoneme based speech recognition development testbed. This has included development in a number of areas to improve system performance in general, and development in a number of areas to make the system more appropriate for NASA's specific requirements. Applications with requirements similar to NASA's, those requiring accuracy under conditions of moderate stress and noise, have also benefited from these improvements.

The Phase II research was divided into three primary tasks. The first task was the development of a system tailored for NASA. Since NASA's requirements are not unique among SSI's potential customer application areas, this development has improved the performance, versatility, and applicability of SSI's speech recognition system in general as well. This task included research and development in the areas of dialect sensitivity, using a headset input device, and speech model development for NASA personnel. This effort has produced a headset system that is now SSI's standard input device. It has also produced a method, called profiling, which allows us to adapt generic speaker models to new speech environments, including new applications, new speakers, new dialects, etc. (within reason).

The second task was work directed at improving system performance in general by making improvements to the basic technology. This task included research and development in the areas of parallel processing for the Phonetic Decoder, and improved accuracy in the presence of noise. This effort has resulted in performance improvements primarily by reducing the decoding time and increasing the recognition accuracy, especially in noise.

The third task was a consolidation of the results and a delivery to NASA of the final improved system.

The following table outlines the research and development topics. This is also an outline of the rest of this final report. In the remainder of this introductory section an overview with the major results of each area will be given. In the remainder of this final report each topic will have its own section, giving in detail the results of all of the research.

NASA Specific Requirements
- Profiler/Adaptation Methods
- Generic Speaker Model Improvements
- Dialect Development (and Custom Models)
- Headset Development

General Performance Improvement
- Parallel Processing in the Phonetic Decoder
  - Decoder Reengineering for Parallelism
  - Predicted Performance of Parallel Implementation
- Noise Sensitivity Reduction/Acoustic Processing
  - Noise Added vs. Clean Tests and Improvements
  - Constant vs. Pitch Synchronous Framing Rate

Acoustic Parameter Smoothing
New Acoustic Parameterization Scheme
Consolidation: Delivery and Installation

1.2. NASA Specific Requirements

At the start of the Phase II contract, there were three primary areas of interest for NASA's applications. These were custom models for NASA speakers, a system that performs as well for the Southern Dialect of English as the current system does for General American English, and a headset input device. Custom models were desired to test the maximum performance possible with SSI's technology. The Southern dialect was needed for any potential NASA speakers of that dialect that would not have custom models prepared for them. The headset input device was more appropriate for NASA's primary application than the handset input device in use at the start of the contract.

A method of building customized speaker models from generic speaker models has been developed during the course of the contract. This allows us to build better speaker models, and to build them more easily. This method is called profiling. It is different from the method called profiling in the original Phase II project proposal.

With the profiler, producing good new customized speaker models takes a somewhat different approach from the former method of building these models from scratch. Generic multi-speaker models are built from a large amount of training data. These generic models are then adapted via the profiler to a new speech environment, which may be a new speaker, dialect, application, level of background noise, or something else. Profiled models are typically better than custom models, because they have the benefit of all the speakers that went into the generic model.

The profiler has been used in much of the work throughout the contract, as well as being developed under the contract. In particular, the profiler was used in producing the custom models for NASA, the generic Southern dialect models, and some noise robustness test models.

We start the section on NASA Specific Requirements by describing the profiler. Then an overview of the progress with generic model building is given, as one of the generic models is the starting point for the profile models. Next, the NASA custom models and Southern dialect development are covered in a section on dialect development. Finally, a section covering the development of the headset input device is given.

Profiler/Adaptation Methods

Since many of the tasks for the project required adaptation or building of speech models for a new speech environment, investigations were made into how best to do this adaptation automatically. The result of this research was a new profiler. The profiler takes speech data from some new speech environment (with known text) and adapts a bootstrap model (usually a generic model) to the new environment. This adaptation method usually provides significant system performance improvements (varying by application) over the bootstrap model. It is often the choice to use for adapting to a new speech environment without extensive data collection and retraining.

Generic Speaker Model Improvements

The generic speaker models (male and female) are typically used as the bootstrap models for the profiler. A review of the performance improvements of the speaker models of the recent releases (3.01 to 3.3) is included to show that the profiler's starting point has also been improving.

Dialect Development (and Custom Models)

Using data collected from eight NASA JSC individuals, an analysis of SSI's speech recognition system on the Southern dialect of English was made. Dialect adaptation was attempted through two primary methods, profiling and a dialect specific dictionary. A combination of the two methods was also tested. As a result, custom (profiled) models were built for all of the NASA speakers collected. Two generic Southern dialect models, male and female, were also built from the NASA data. The generic models provide a suitable bootstrap model for any profiling NASA may want to do later for new speakers, if the generic performance is not yet sufficient. The new models have all shown performance improvements in tests with NASA's primary application, a computer aided instruction application for training of mission control specialists.

Headset Development

A headset input device was requested by NASA and many of SSI's other customers. A standard headset input device was chosen for SSI's system. This headset has now become our standard input device. The system performs better with the headset than with the handset, due to a variety of microphone issues. All releases since release 3.0 use the headset input device.

1.3. General Performance Improvements

Two areas were chosen for investigation for general performance improvements of interest to NASA. These were using parallel processing in the Phonetic Decoder, primarily for speed improvements, and improving the recognition accuracy in the presence of noise.

Parallel Processing in the Phonetic Decoder

Implementing an algorithm in parallel and porting to a parallel processor is one way to improve the processing time of an algorithm, if the algorithm is suitable for parallelization. One obvious problem for users of speech recognition systems is the time required to process the input speech. The Phonetic Decoder was analyzed for implementing in parallel as a method of reducing the decoding time. This was done in two steps: reengineering the decoder for parallelism, and predicting the decoding time when ported to a parallel machine.

Decoder Reengineering for Parallelism

In preparation for porting to a parallel machine, many changes were made to the Phonetic Decoder. Most of these resulted in decoding time reductions themselves, somewhat removing the need for speed increases due to a parallel implementation. Since the start of the project, the decoder reengineering changes have produced performance improvements of about 1-5% absolute utterance accuracy and 75-85% decoding time reduction, depending on the application.

Predicted Performance of Parallel Implementation

An analytical model was prepared for predicting the time reduction improvements from porting the reengineered decoder to a parallel processor. Time measurements were made for SSI's "home" computing environment, an ethernet network of SUN workstations. This analysis showed that porting to a parallel implementation in this environment would not improve execution time, primarily because of the communication overhead of the network. However, an improvement in the communication overhead time of one order of magnitude would provide a system that would speed up in a parallel implementation. Such a change is not unreasonable to expect, given that our "home" system is relatively loosely interconnected.

Noise Sensitivity Reduction/Acoustic Processing

In the Phase I study we learned that noise in the simulated mission control room at NASA JSC did have an impact on recognition accuracy. An investigation was made into the nature of this impact, and several methods were implemented to improve the noise robustness of the speech recognition system.

Noise Added vs. Clean Tests and Improvements

This task studied the impact of algorithmically added noise on the recognition accuracy. Profiling was examined as a means of adapting to noisy speech environments. The system was found to be fairly robust with respect to low to medium noise levels. Profiling was found to recover most of the lost accuracy under noisy conditions.

Constant vs. Pitch Synchronous Framing Rate

SSI's speech recognition system originally used a pitch synchronous framing rate for the initial segmentation of the input speech data. A constant framing rate was implemented for a number of reasons. This eliminates some of the variability introduced by the pitch tracker in noisy environments. A constant rate is required for testing some alternate acoustic processing methods. Having a constant rate for all models makes the profiling process easier, as this rate does not have to be adapted for new speech environments. In our tests, constant framing rate models produced similar performance to pitch synchronous models.

Acoustic Parameter Smoothing

A smoothing method for the acoustic parameters produced by the acoustic processor was implemented after constant frame rate processing was available. The smoothing primarily improved decoding time, by producing cleaner input to the later stages of the speech recognition system. The smoothing can be seen to remove some of the random noise of the speech signal.

New Acoustic Parameterization Scheme

An alternate acoustic parameterization of the speech input was tested and compared with SSI's standard acoustic parameterization. The standard scheme uses a filterbank method; the alternate we tested used a cepstrum parameterization. Our results showed that both schemes yielded essentially equivalent speech recognition performance. From this we conclude that there is no loss in retaining the current acoustic parameterization scheme.

1.4. Consolidation: Delivery and Installation

A consolidation of the improved speech recognition system was built. The improved system included the NASA specific speech models (including better recognition of the Southern dialect), changes to the system from the parallel processing study and the noise study for improved speed and accuracy, the headset input device, and changes from the most recent release 3.3.1 from other internal research.

The final system was delivered and installed at NASA JSC. The new system showed improved performance in all important ways: speed, accuracy, noise robustness, and Southern dialect recognition.

SSI's ability to deliver this improved system, and our increased understanding of how to better prepare for applications in high stress, moderate noise environments, provides a vehicle for the Phase III commercialization of the technology developed under this study. This addresses that objective of the SBIR program in funding this research.

1.5. Conclusions

We obtained significant improvements in performance, for both speed and accuracy, as a result of this study. We have also obtained significant improvements in our technology for adapting to new speech environments. The results of our investigations will improve the utility of SSI's commercial system and provide technology and insights for other developers of speech recognition systems.

The study has shown and delivered a phoneme based speech recognition system that operates successfully in high stress, moderate noise environments. This system is suitable for mission control training and other real applications at NASA, and wherever robust, natural, fast and accurate speech input for computer systems is required.

2. NASA SPECIFIC REQUIREMENTS

2.1. PHONETIC PROFILING

Adapting a Generic Speech Recognition Model to New Conditions

Abstract

In this section, we discuss phonetic profiling. This approach was originally developed in order to deal with the shortcomings in the speaker coverage of some of our generic speech recognition models. While since then the speaker coverage of the generic models has improved significantly, we have been able to apply the phonetic profiling approach to many scenarios which require adapting these models to conditions with different acoustic and/or phonetic characteristics. In this section, we will address three major points:

- Why is there a need for adaptation?
- What are some possible applications of phonetic profiling?
- What is involved in the process of phonetic profiling?

2.1.1. Need for an Adaptation Approach

A speech recognition model must deal with many variables such as acoustic characteristics of the speakers (physical characteristics of the vocal apparatus and dialect, for example), the recognition environment (high noise level, for instance), and the application vocabulary and word context. Accounting for all the variabilities in the above parameters in building a truly generic speech recognition model will most likely results in a less than optimal model for any given application scenario. Furthermore, the task of creating a generic speech recognition model requires collecting a large amount of data. This is a very time-consuming and expensive process. Therefore, it can be difficult to create an entirely new model for a new application, recognition environment, or group of speakers.

Given our generic speech recognition models, phonetic profiling is an attractive alternative. Generic speaker models are built from a large and diverse set of utterances from many speakers using a broad vocabulary. While these models provide us with a reasonable level of recognition performance robustness, they also yield themselves very well to adaptation to significantly different conditions. Phonetic profiling is an extremely flexible technique for adapting these models in order to achieve the best performance for a given set of recognition parameters.

2.1.2. Applications of the Phonetic Profiling Technique

As we mentioned earlier, a speech recognition model must deal with different types of acoustic and phonetic parameters which characterize a given recognition scenario or application. Any significant change in one or more of these parameters may necessitate the need for profiling the generic speaker models in order to achieve the best recognition accuracy without having to perform an extensive retraining. Some of the areas to which we have applied the profiling approach include:

- Adapting a model to a speaker with poor recognition performance. Factors contributing to the poor performance of a speaker on a generic model could include acoustic speaker characteristics that are very different from those of the training speakers. The physical characteristics of the vocal apparatus or the heavy dialect of a speaker can be mentioned as some of the examples.

- Adapting a model to specific conditions of a recognition environment. Significantly high levels of noise in the environment can have an adverse effect on the recognition performance of models trained on data collected under normal conditions. The recognition accuracy of these models can be improved by profiling them on data collected in the application environment.

- Adapting a model to a specific application vocabulary. An application may be rather unique in terms of its phone or word context.

In several parts of this project report, sections on noise and dialect study for example, we have presented the results for different profiling scenarios. For example, we were able to achieve a word error rate reduction of about 52% when we profiled a generic female model for a NASA female speaker with noticeable southern dialect. The test syntax was one of our NASA applications and most of the speech data used for profiling was based on this same application. As you can see from these results, profiling very often results in a slow-down in decoding; however, the accuracy improvements justify the slowdown which is hardly noticed by a user during recognition.

Test Set = About 200 utterances.

Note on reading the tables:

"Word Acc"	word accuracy as a percentage of all words spoken.
"WTrans	word accuracy as a percentage of all words transcribed.
"Utt Acc"	percentage of sentences decoded with no errors.
"+ Off 1"	percentage of sentences with one or less errors.
"+ Off 2"	percentage of sentences with two or less errors.
Segs/Utt	average number of phonetic segments per utterance.
"CPU/Utt "	average SUN SparcStation 1 decoding time per utterance (in seconds).

Model	*WordAcc*	*WTrans*	*UttAcc*	*+Off1*	*+Off2*	*Segs/Utt*	*CPU/Utt*
generic	90.56	91.18	74.24	81.82	91.41	42.87	0.47
prof1	95.45	95.53	81.31	93.43	94.95	43.95	0.89

In order to evaluate the impact of noise on the recognition accuracy and study profiling as a possible means of dealing with it, we added collected environment noise (significantly boosted) to the above sets of profiling and testing speech data. You can see that the best accuracy on the noisy data is achieved when we use a model profiled on data containing same level of noise (i.e. prof2):

Model	*WordAcc*	*WTrans*	*UttAcc*	*+Off1*	*+Off2*	*Segs/Utt*	*CPU/Utt*
generic	80.79	84.20	58.59	70.71	83.33	55.67	1.02
prof1	88.12	90.41	68.69	81.82	89.90	57.21	1.62
prof2	92.75	92.75	74.75	87.37	92.93	59.24	1.32

2.1.3. The Profiling Process

In order to better understand the profiling approach, we present a brief overview of the steps involved in the recognition process. There are three main stages:

1) Acoustic Processing

A spectral analysis of the digitized speech data is performed and each utterance is represented by a sequence of 6.6 millisecond frames of speech. A vector of 24 acoustic parameters is associated with each frame.

2) Phonetic Encoding

This stage consists of several steps:

a) Each frame of speech data is run down a first-stage frame-level classification tree trained on labelled speech data.

b) A mapping of the frame tree's terminal nodes to broad phonetic classes (frame tree terminal node map, TNM) is used to represent each utterance as a sequence of broad phonetic classes. Segments are formed based on runs of these classes.

c) A broad phonetic class likelihood (given nodes of the frame tree) matrix is used to remove the spurious segments and form "clean" runs of broad classes. Now each utterance is represented as a sequence of these segments instead of the 6.6 ms frames.

d) Each segment is run down a second stage segment-level classification tree. This tree can examine wider context and use categorical information in making its classification decisions. The output of this tree is a sequence of phonetic codes which represent the utterance.

3) Phonetic Decoding

A phonetic codebook (PCB) containing information about the phonetic codes representing each utterance and the phonetic classes used for spelling words in the dictionary is used to decode the speech.

A speech recognition model contains all the components built during the training process to perform the above tasks. For a typical custom profiling process, adaptation based on speech data from a single speaker, the user is normally required to collect about 40 minutes of speech defined by a given text. This text should normally be representative of the user's application. On the average, the profiling process on this amount of data takes about two hours on a SUN SparcStation1.

Given the text of the collected speech and a dictionary containing phonetic spellings of the vocabulary, the profiler labels every frame of speech and changes three components of the speech recognition model: the frame-level terminal node map (TNM), the class probability matrix used for segmentation cleanup, and the phonetic codebook (PCB). The term "phonetic profiling" is therefore used to indicate a change in the phonetic components of the speaker models.

Depending on the nature of a given condition to which a model is to be adapted (and how different it is from the original training conditions), we sometimes consider a less extensive form of profiling which is referred to *PCB augmentation*. This process results in a change in only the PCB component of the speaker model. The change is due to the addition of new data to the phonetic codebook. The original training data still take part in generating the likelihood scores in the PCB.

2.1.4. Conclusion

A reasonable level of recognition performance robustness with respect to speaker and environment variations should be achieved before considering an adaptation approach. An adaptation technique can then be utilized as an inexpensive approach to achieve the best recognition accuracy performance under new conditions (with significantly different characteristics from the original training conditions) without extensive retraining.

2.2. GENERIC HEADSET SPEAKER MODEL IMPROVEMENTS

Releases 3.01 to 3.3

2.2.1. Overview

With the implementation of the profiler, most of our model building efforts are expended in building generic models. The generic models form the basis for profiling by providing the "bootstrap" models for adaptation to a specific user, users, applications, etc. The adapted models then have the benefits of both (a) adaptation for improved recognition for a specific application, and (b) lots of data from diverse speakers and applications for good recognition in general.

In this section we present the improvements in the generic models of SSI's general releases over the course of the contract. These start with release 3.01, the first major headset and PE200 release. These results are included to give the baseline performance on independent speakers. We start from here; speaker model adaptation (e.g., profiling) then improves the performance for a specific speaker, speakers, application, or whatever new data for which we want to customize.

2.2.2. Test Results

Every major general release includes (at least) new generic male and female general application speech models. In this section, we give the performance results of these models for the last four major releases: 3.01, 3.1, 3.2, and 3.3.

The performance is evaluated on two tests: and pmtext test and the digits test. The digits test uses a grammar accepting any number, any order of digits in an utterance. This is an unrestricted grammar on the digits vocabulary. The pmtext (performance measurement) test uses a grammar developed from a list of diverse vocabulary utterances, which was then generalized by adding confusing words to word categories and allowing for inserted and deleted word categories. This is a test designed to be difficult so that we can see even small changes in performance, and so that we are not saturated at either the high or low end of performance.

For the earlier releases, 3.01 and 3.1, four models were delivered: male and female generic models for general applications and digits applications. For release 3.2 and 3.3, there were only the generic male and female general application models (no separate digits model). The digits tests below for releases 3.01 and 3.1 are on different models than the pmtext tests, but the digits and pmtext tests use the same model for the release 3.2 and 3.3 tests.

Normally we make it a point in our tests to change as few inputs to the system as possible, to evaluate individual differences in a consistent setting. These results are of the other extreme: very little is the same across all of these tests. The grammars are the same. The tests all use data from at least four speakers, all of which are independent of the model building speaker set. However, the actual independent speakers used may be different, as some speakers independent of earlier releases were dependent for later releases, so new independent speakers were required for testing.

Each test uses the best technology available at the time of the release. The speed/accuracy parameter tradeoff point chosen for a release was used with the tests of that release. The

dictionaries, coarticulation methods, and the phonetic decoder, as well as the model building technology are all the best versions corresponding to the given release. Thus although these results are not comparable in our usual sense, they are comparable in the sense that they represent an overall evaluation of the technology of a given release.

pmtext male

release	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
3.01	81.15	81.81	32.15	62.62	77.85	19.01
3.1	83.51	84.14	30.77	62.92	81.08	16.96
3.2	85.28	87.56	36.36	71.19	86.29	20.73
3.3	88.58	90.52	36.80	73.80	89.98	10.27

digits male

release	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
3.01	89.39	85.95	56.46	82.88	94.19	0.68
3.1	90.24	86.56	55.86	84.18	94.99	0.45
3.2	87.32	83.70	50.86	79.52	91.22	0.37
3.3	92.48	93.73	71.59	91.79	97.85	0.39

pmtext female

release	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
3.01	82.02	84.90	28.92	62.31	81.23	15.23
3.1	84.24	86.93	29.23	65.08	83.38	14.60
3.2	82.24	84.74	32.82	68.57	81.20	19.69
3.3	88.10	88.46	39.11	68.98	88.44	10.48

digits female

release	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
3.01	91.92	87.02	60.75	84.71	95.08	0.46
3.1	92.29	89.50	64.96	88.69	96.30	0.44
3.2	90.37	86.14	59.40	83.22	93.07	0.42
3.3	92.53	89.71	66.08	88.82	96.11	0.47

These results show that the accuracy and decoding time performance measurements of the generic models, with minor exceptions, have improved with each release. Sometimes decoding time has been traded-off for improved accuracy, and vice versa, but overall improvements have come in both areas. The best improvements have come with our most recent major release, 3.3.

As the performance of the bootstrap generic models improve, so does the performance of the adapted models based on them. The profiler and other adaptation methods allow customized models to inherit the improvements of the generic models. Elsewhere we have shown how the adaptation methods improve performance on new speakers and applications. The results here show that the recognition starting point, the performance on the baseline bootstrap generic models, has also been improving with each release.

2.2.3. NASA Specific Deliverable Models

The profiler has been used to prepare the NASA specific models required for delivery in our contract. The bootstrap models used for profiling were the original generic male and female general release 3.3 models. Profiling was used instead of building custom models, since profiling produces better results. This happens because a profiled model has the benefit of all the other data that went into building the original generic model.

Ten models have been prepared for the final NASA specific deliverables. These include eight individual speaker models, four male and four female. The two other models are generic southern dialect models, one male and one female. All of these models are headset models.

Detailed building and performance reports for these models are included in the next section, on dialect development. These models fulfill the model building requirements of the contract, as well as providing major performance improvements for NASA's applications.

2.2.4. Conclusions

All of the major releases of SSI's technology have produced performance improvements for the general release generic models. Profiling and other adaptation methods have been made available so that speech models can be customized to new speakers and/or applications. This allows them to inherit the benefits of the new generic models, while giving improvements for the new speakers and/or applications. The NASA specific deliverable speech models have been built on the most recent (3.3) general release generic models, containing the best of our research to date.

2.3. DIALECT DEVELOPMENT

2.3.1. Introduction

Two principal questions are addressed in this section:

(1) Does profiling improve performance for dialect speakers?
(2) Is a dialect sensitive dictionary necessary for adequate performance?

The answer to the first question is in the affirmative. The answer to the second question is in the negative.

This report covers the initial recognition results for the principal NASA commands grammar and the integers grammar for custom and gang profiles using the baseline release 3.3 generic (3008, 3009) models and profiled models. The grammar most related to a practical application was the NASA commands one. This grammar is from a NASA computer aided instruction (CAI) application for mission control specialists.

The mean word accuracy recognition improvement for the NASA commands for custom profiles with the standard dictionary was 2.48% for an error reduction of 31.96%. The performance of the release 3.3 standard dictionary profiles was marginally poorer than the multiple-transcription dictionary (0.26% lower word accuracy; 3.72% error increase). This

result did not appear to justify using a multiple-transcription dictionary for dialect speakers. Also the speakers with the strongest dialect received the least benefit from a multiple-transcription dictionary, and females tended to be better recognized with the standard dictionary.

2.3.2. Problem and Methodology

Standard speech models (male, female) are built with speakers whose speech can be described as General American (GA) with a West Coast bias. Speakers of other dialects of English tend not to perform as well on these models as General American (GA) speakers, all other factors being equal (speaking rate, phone deletions, age).

The task of speech recognition is to convert speech (variations in air pressure carrying information) into equivalent strings of digital characters. Usually the output strings are regular English text when represented on an output device. Only a portion of the information carried by the pressure wave deals with that needed for output. One fundamental problem in speech recognition is that different pressure wave sequences can represent the same output while, conversely, rather similar sequences can represent different output.

The systematic variation in speech can be attributed to several interacting sources. Physically the pressure waves are affected by the physiology of the particular vocal tract producing them. For simplicity, one can think of the tube between the vocal cords and the mouth as a horn. The sound of the horn varies as one puts the tongue in different locations. A good introduction to this can be found in [Ladefoged 71] or [Borden and Harris 84].

Behaviorally, the pressure waves are affected by the learned patterns of tongue placement (and other articulator movement). These behavioral patterns are traditionally related to concepts such as idiolect, dialect, style and register which are delineated in [Hudson 80].

The speech of NASA commands can be classified as an occupational register with a somewhat formal style associated in the current study with at least three identifiable dialects and an idiolect for each person. Register identifies the characteristics of NASA commands which are unique to that organization. These differences are primarily found at the lexical level and pose no significant problems for speech recognition. Style identifies the characteristics due to the relationships obtaining among the interlocutors, somewhere between the intimacy of married couples and the distance of a formal scientific presentation at a conference. Dialect accounts for the variation due to regional and cultural influences upon a person's speech. One can identify characteristics typical of the Southwest region and Hispanic culture. The term General American (GA) is a somewhat abstract dialect representing the common features of the speech of most of the regions of the United States in a fairly formal style. One would expect that the somewhat formal character and national orientation of speech within NASA would cause speakers to alter their speech in the direction of GA. Idiolect refers to the peculiarities of an individual's speech.

The speech recognition system deals with the problem of different vocal tract sizes by using two models (male for large vocal tracts, and female for smaller vocal tracts). The system deals with other individual differences by building each model on the data from several speakers. Heretofore, no systematic effort has been made to model the differences due to dialect. An example of this difficulty is the pronunciation of the word 'nine.' In GA this would be transcribed as /nayn/ whereas in the Southern dialect it would be transcribed as /na:n/. The phonetic dictionary in the system has the first transcription, but the system

will always output the best match it can find. If there is a word in GA which differs from another only by the vowels /ay/ and /a:/, the system would likely make the wrong choice for a Southern speaker provided that both words can occur in the same syntactic position. For the case of NASA commands these situations are relatively rare, e.g. 'plight' and 'plot' might be confused without a syntax, but would not be if the syntax required that only one of the words be decoded as a verb.

Methods for Adapting the System

Three methods can be used to improve the recognition of the system for a particular speaker or group of speakers. First, the system can be trained to identify the speaker's speech patterns with existing word transcriptions (via profiling). Second, word transcriptions can be changed to be more like those of the speaker's speech patterns by modifying the dictionary. The third method, pcb augmentation, is somewhat intermediate since it involves decoding words with existing models and transcriptions, but changes the codebook to reflect the characteristics of the new data. Profiling and pcb augmentation were discussed earlier in section 2.1.

Experience has shown that the first method (profiling) tends to work better for most situations for two reasons. One, the speech characteristics of the speaker are modelled, and, two, additional data for particular words are added to the model. Additionally, if the data are added to an existing model (profiling) rather than used to build a model only for one speaker, recognition is usually the same or better for other speakers. PCB augmentation is often as good or better than profiling if the data are markedly different from that used to create the original model.

Adding transcriptions to a dictionary tends to only be useful if speakers consistently use two different pronunciations for the same word (as for example 'the' which can be transcribed as /ðə/ or /ði:/), and at least one of the pronunciations can be easily confused with another word. Confusibility tends to be higher for shorter words such as 'the.' The wholesale addition of multiple pronunciations did not seem to benefit speakers of GA, but had not been tried for dialect speakers.

2.3.3. Test Setup Details

In this study, four male and four female speakers representing a continuum from GA to Southern (South-Western) contributed speech. All of the speakers were NASA staff from the Johnson Space Center. There is no record of speech or hearing defects.

Each speaker contributed 2019 utterances. For the dialect phase of the investigation, 1600 utterances were used per speaker. However, part of the original data for one speaker was corrupted and irrecoverable. A small percentage of utterances for each speaker failed quality assurance tests and were also not used for model profiling, but the test sets were identical for all speakers, i.e., the same number of test utterances were used for each speaker.

The data sets were divided by speaker, type of utterance, and whether the data was used for profiling or testing. For each speaker, there were 200 test utterances of the new NASA type (commands), 100 test utterances of the integer type, and 75 test utterances of the phonetically balanced "phobal" type. The NASA utterances had commands such as, "Move vector to slot v 10." The integer type consisted of integers such as "five," "twenty," and "point" in various combinations. The phobal type had the set of English phonemes in different contexts. The first two types are useful for NASA applications while the last

serves to populate any categories in the profile which are inadequately represented by the other two types.

Original determinations of dialect intensity and type were made by the SSI staff who collected the data. These impressions were subsequently confirmed by another staff person, with graduate work in phonetics, auditing four utterances from each speaker.

In general, it can be noted that speakers appeared to be using a more standard form of pronunciation than that which might have been achieved in less formal circumstances. However, the recording circumstances are likely to represent the actual variety used by NASA personnel using a speech recognition system. In other words, a person speaking to a computer using NASA jargon, is likely to use a pronunciation closer to that of GA than he or she might use with friends or relatives in an informal gathering. This effect may help to account for a smaller degree of spread than was initially anticipated.

It was still the case that the speakers with the strongest dialects had marked departures from GA. These speakers also tended to speak slightly more slowly than normal, although the difference could not be statistically verified due to the small sample size. Slower speakers (within reason) tend to be better understood by the recognition system.

From the above discussion, it should be apparent that under NASA conditions it should be possible to achieve fairly typical recognition results for speakers of Southern dialect, and that speakers with only a slight divergence from GA should not be extremely different from speakers for which no Southern accent could be detected.

Technical details of the recording methods and acoustic processing have been detailed elsewhere (NASA Phase II: Interim Report 2).

Speaker Characteristics

The speakers will be identified by first names: bob, bowen, bryan, james, carla, gloria, jacque, and pam. The characteristics of the speakers are as follows:

- bob: strong Southern dialect, low-mid pitched register, middle-aged.
- bowen: moderate Southern dialect, low pitched register, early middle-aged.
- bryan: very little Southern dialect, mid-high pitched register, early twenties.
- james: very little Southern dialect, middle register, mid-thirties.
- carla: moderate Southern dialect, low-mid register, mid-twenties.
- jacque: moderately strong Southern dialect, middle register, mid-twenties.
- gloria: very little Southern dialect, some Hispanic accent, low-mid register, middle-aged.
- pam: little Southern dialect, middle register, mid-twenties.

Chief characteristics were that the /aɪ/ diphthong of GA was monopthongized by the stronger dialect speakers. There were also slight changes in the quality of other vowels, especially /æ/ to /ɪ/ in some cases. The stress patterns also were slightly different from GA for the stronger dialect speakers. Rate and loudness factors appeared to be within normal range for all speakers.

Test Dictionaries and Grammars

Two dictionaries were used for testing and three grammars. The two dictionaries tested one of the research questions. The three grammars were needed to decode the three types of text used in the test data: NASA commands, integers, and phonetic material.

The first dictionary was the general release 3.3, standard dictionary of June 21, 1990. This dictionary had one GA phonetic transcription per word. It was used to create and test the custom profiles. This dictionary along with the appropriate standard settings would give the baseline performance of the system without any special accommodation to the particular speaker or group of speakers.

For the gang profiles the general release 3.3.1 dictionary was used (September 19, 1990). This dictionary had an extra transcription for 'the,' but otherwise was the same for the vocabulary tested.

The alternate dictionary was based upon an old standard dictionary which had multiple transcriptions which were more sensitive to dialectal effects such as /r/ dropping and cluster reductions. Its use was discontinued for the general release after it was found that these alternatives did not significantly improve accuracy for GA speakers and the extra transcriptions resulted in slower processing speeds. It was not known whether the extra options would be useful for dialect speakers, but since many of the transcriptions had been included in order to aid dialect speakers, it was decided that this dictionary would provide a good contrast with the current 3.3 general release dictionary. This dictionary will be referred to as the NASA_dialect dictionary. In the event that this dictionary had shown better performance than the standard, an even more dialect specific dictionary would have been attempted. Actually some such testing was performed on the speaker with the strongest dialect, but little or no improvement was noted. Those tests were performed with single (dialect specific) transcriptions.

The three grammars specified permissible sentences for the vocabulary tested. The first grammar (referred to as NASA_only) was a typical grammar which constrains the sequence of words to those which are grammatically acceptable ('the book is big' is possible; 'big book the is' is not possible). A listing of this grammar can be found in Appendix 1 of NASA SBIR Phase II Proposal: A Phoneme Based Speech Recognition System for High Stress, Moderate Noise Environments. The other two grammars permitted any integer or any word to follow another. These are referred to as the infinite_integer and infinite_phobal (PHOnetically BALanced) grammars. Since these are syntactically unconstrained, the performance is typically much worse than for grammars which use syntactic restraints.

Models Used in Testing

For recognition testing the generic models were used to establish baseline performance. Profiled models, based upon either standard dictionary (STD) labelled data or NASA_dialect dictionary (MTD) labelled data, were also tested. The standard dictionary based profiles will be designated by the bootstrap generic model number (3008 or 3009, for female or male) plus the word 'single' (for single transcription dictionary, STD). Models based on the NASA_dialect dictionary are identified by the affix 'mult' for multiple-transcription dictionary (MTD). The total number of models is 24: 2 baseline generic, 8 custom STD (one per speaker), 8 custom MTD (one per speaker), one female gang profile (STD), one male gang profile (STD), two female gang pcb augmentations (different augmentation weights, STD) and two male gang pcb augmentations (different augmentation weights, STD). "Gang" profiling or augmentation refers to adapting a model on several speakers' data together, to build a multi-speaker model.

Recognition Settings

The decoding program which attempts to match code sequences from the model with acceptable words and sentences was set to normal settings except for a few runs where options such as no jumping were tested. The normal settings used were slider setting 6, jumping on and constant language weight 0. Times reported are for a Sun 4/110 workstation. Slider setting six sets various thresholds at their maximum accuracy for that release. Jumping permits codes to be matched with more than one phonetic class. Some tests were done where this feature was not allowed when experience suggested that better results might be achieved without it. The constant language weight of zero means that no attempt was made to decrease word scores by a constant which serves to reduce false insertions. Such calibration is highly syntax dependent.

2.3.4. Results

The results of the various tests are presented in this subsection. The results are given by profile type (custom or gang), by application text type (mission control specialist CAI application commands or integers), and by model gender (male or female).

Three types of text were used: NASA commands, integers and phonetically balanced nonsense phrases (phobal). Only the results for the commands and integers are shown for the custom profiles since these are directly comparable to the types of practical applications which would be used by NASA. The summary recognition results for both texts are presented females first followed by males. For the commands, individual speaker results are also given: baseline followed by single transcription profile followed by multiple transcription profile, each tested on standard (single transcription) and multiple transcription dictionaries.

Table labels indicate the model and dictionary used as well as the word accuracy, percentage of words transcribed, utterance accuracy, percentage of utterances with one error, percentage of utterances with two errors, average number of segments (codes) per utterance, and average decoding times. The average decoding times are for a Sun 4/110 and usually the decoding was done in the background. Actual speech recognition is performed with a dedicated processor, but the values here will indicate general trends vis a vis the baseline.

For the females the following number of utterances passed quality assurance tests and were used to profile the generic models : carla 1129; jacque 1143; gloria 1148; pam 1150.

For the males the number of utterances used for profiling were : james 1149; bob 1149; bowen 923; bryan 1150. Note that one recording (~13 sessions) of bowen was bad, and could not be recovered.

The profiling text used to build the models consisted of primarily NASA commands (800 utterances) with the remainder consisting of integers and phonetically balanced material.

Profile Type: Custom; Application: NASA Commands; Gender: Female

These results were all obtained using the NASA commands application grammar and the 'nasatest' collected text which had 200 utterances in it. A few utterances were not well recognized due to insufficient leading and training silence, which is usually due to speaker error. Such errors tend to decline with practice and represent a very small proportion of the test utterances for each subject (around 1%).

Summary of Results:

A. Overall Results: Profiled vs. Non-profiled Tested with the Single-Transcription Dictionary

In this section the table indicates the performance increase and percentage of error reduced for each speaker for the single-transcription (STD) model over the release 3.3 generic female model (3008).

Table I. Improvement: Single-Transcription Profile vs. Baseline Both Tested with Single-Transcription Dictionary

Speaker	carla	gloria	jacque	pam	MEAN
Word Accuracy (%)					
Recog. increase	2.42	2.42	5.43	1.25	2.88
Error reduction	40.27	46.76	58.07	18.71	40.95
Utt Accuracy (%)					
Recog. increase	0.50	6.00	8.00	2.00	4.12
Error reduction	3.12	30.00	32.00	8.89	18.50

Note: Error reduction = [speaker_recog_increase / (100 - speaker_baseline_correct)] times 100 to convert proportion to percent.

B. Summary of Results for Each Speaker:

This section presents the word accuracy, word transcribed, utterance accuracy, segments per utterance means and mean decoding time for each speaker as defined above. The models and dictionaries used are indicated in the table names.

Table II. Baseline 3008 Model Tested with Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
carla	93.99	96.08	84.00	92.50	97.50	44.48	0.63
gloria	94.74	95.86	80.00	91.00	97.00	41.04	0.74
jacque	90.65	91.72	75.00	82.00	91.50	42.76	0.71
pam	93.32	94.83	77.50	89.50	95.50	41.26	0.68
MEAN	93.18	94.62	79.12	88.75	95.38	42.38	0.69

Table III. Profiled 3008_single Models Tested with the Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
carla	96.41	96.41	84.50	93.00	98.50	40.99	1.26
gloria	97.16	96.52	86.00	96.00	97.00	39.31	1.12
jacque	96.08	96.08	83.00	93.00	97.00	43.79	1.54
pam	94.57	95.05	79.50	90.50	96.50	39.12	1.50
MEAN	96.06	96.02	83.25	93.12	97.25	40.80	1.36

Table IV. Profiled 3008_mult Models Tested with the Multiple-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
carla	96.58	96.50	84.00	94.00	98.50	41.35	1.76
gloria	97.08	96.67	85.50	96.00	97.50	39.26	1.54
jacque	95.66	94.71	80.00	91.00	96.50	43.88	1.91
pam	94.32	94.56	77.00	89.00	97.50	39.13	1.94
MEAN	95.91	95.61	81.62	92.50	97.50	40.90	1.79

Table V. Profiled 3008_single Models Tested with the Multiple-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
carla	96.24	95.61	84.50	92.00	97.50	40.99	1.58
gloria	97.58	97.25	88.50	96.50	97.00	39.31	1.63
jacque	96.16	95.05	81.50	90.50	97.00	43.79	1.72
pam	95.16	95.64	80.50	92.00	96.00	39.08	1.58
MEAN	96.28	95.89	83.75	92.75	96.88	40.79	1.87

Table VI. Profiled 3008_mult Models Tested with the Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
carla	96.83	97.15	84.50	96.00	99.00	41.35	1.14
gloria	96.91	96.83	85.50	95.50	97.50	39.26	1.06
jacque	95.74	95.50	81.00	93.00	97.00	43.88	1.39
pam	93.91	94.30	75.50	89.00	97.50	39.17	1.36
MEAN	95.85	95.94	81.62	93.37	97.75	40.92	1.24

C. Difference Between Profiling with Two Types of Dictionaries

This section compares the two profiled models for each speaker. A positive entry indicates that the single-transcription (STD) model was superior to the multiple-transcription (MTD) model. The STD results used the single-transcription dictionary for both profiling and testing the MTD results used the NASA_dialect dictionary (MTD) for profiling and testing.

Table VII: Improvement: Single-Transcription Over Multiple-Transcription Profiles Tested with Single- and Multiple-Transcription Dictionaries Respectively

Speaker	carla	gloria	jacque	pam	MEAN
Word Accuracy (%)					
Recog. increase	-0.17	0.08	0.42	0.25	0.14
Error reduction	-4.97	2.74	9.68	4.40	2.96
Utt Accuracy (%)					
Recog. increase	0.50	0.50	3.00	2.50	1.62
Error reduction	3.12	3.45	15.00	10.87	8.11

Profile Type: Custom; Application: NASA Commands; Gender: Male

The male tests used the same grammar and text as the female results. The generic model was 3009, standard release 3.3 model. The tape damage which limited the number of

utterances available for profiling bowen also had the effect that bowen's tests had only 150 utterances in them rather than 200.

IV. Summary of Results

A. Overall Results: Profiled vs. Non-profiled Using Single-Transcription Dictionary

In this section the table indicates the performance increase and percentage of error reduced for each speaker for the single-transcription (STD) model over the generic model (3009).

Table I. Improvement: Single-transcription Profiles Over Baseline

Speakers	bob	bowen	bryan	james	MEAN
Word Accuracy (%)					
Recog. increase	1.67	4.37	2.00	0.25	2.07
Error reduction	27.79	30.24	29.24	4.60	22.97
Utt Accuracy (%)					
Recog. increase	1.50	2.03	1.00	-3.00	0.38
Error reduction	8.33	5.78	5.13	-17.14	0.52

B. Summary of Results for Each Speaker:

This section presents the word accuracy, word transcribed, utterance accuracy, segments per utterance means and mean decoding time for each speaker as defined above. The models and dictionaries used are indicated in the table names.

Table II. Baseline 3009 Model Tested with the Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
bob	93.99	95.34	82.00	91.50	96.00	43.49	0.73
bowen	85.55	90.41	64.86	81.76	90.54	40.03	1.08
bryan	93.16	95.63	80.50	89.00	96.00	42.33	0.74
james	94.57	96.26	82.50	91.50	96.50	42.70	1.04
MEAN	91.82	94.41	77.46	88.44	94.76	42.14	0.9

Table III. Profiled 3009_single Models Tested with the Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
bob	95.66	96.71	83.50	93.50	98.50	43.44	1.33
bowen	89.92	91.04	66.89	82.43	90.54	38.48	1.59
bryan	95.16	96.20	81.50	91.50	98.00	41.44	1.41
james	94.82	95.86	79.50	93.00	96.50	40.72	1.51
MEAN	93.89	94.95	77.85	90.11	95.88	41.02	1.46

Table IV. Profiled 3009_mult Models Tested with the Multiple-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
bob	95.91	96.39	82.50	93.50	98.50	43.45	1.54
bowen	91.83	92.76	72.97	84.46	91.89	38.29	1.72
bryan	95.74	96.22	83.00	91.50	98.50	41.17	1.43
james	94.74	95.70	79.00	91.50	97.00	40.66	2.33
MEAN	94.56	95.27	79.37	90.24	96.47	40.89	1.76

Table V. Profiled 3009_single Models Tested with the Multiple-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
bob	95.83	96.31	82.50	93.00	98.50	43.44	1.56
bowen	90.15	91.58	68.24	82.43	89.86	38.48	1.73
bryan	95.83	96.15	82.50	91.00	98.50	41.44	1.49
james	94.74	95.46	77.50	92.00	97.00	40.72	2.28
MEAN	94.14	94.88	77.68	89.61	95.96	41.02	1.76

Table VI. Profiled 3009_mult Models Tested with the Single-Transcription Dictionary

speaker	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
bob	95.08	96.12	80.50	92.00	98.00	43.45	1.35
bowen	91.15	92.50	71.62	85.14	91.22	38.24	1.63
bryan	95.41	96.21	82.50	92.00	97.50	41.17	1.28
james	94.66	95.86	79.00	92.00	96.50	40.66	1.50
MEAN	94.08	95.17	78.40	90.41	95.80	40.88	1.44

C. Difference Between Profiling with Two Types of Dictionaries

This section compares the two profiled models for each speaker. A positive entry indicates that the single-transcription (STD) model was superior to the multiple-transcription (MTD) model. The STD results used the single-transcription dictionary for both profiling and testing the MTD results used the NASA_dialect dictionary (MTD) for profiling and testing.

Table VII. Single-Transcription Profiles Over Multiple-Transcription Profiles Tested with the Single- and Multiple-Transcription Dictionaries Respectively

Speakers	bob	bowen	bryan	james	MEAN
Word Accuracy (%)					
Recog. increase	-0.25	-1.91	-0.58	0.08	-0.66
Error reduction	-6.11	-23.38	-13.62	1.52	-10.40
Utt Accuracy (%)					
Recog. increase	1.00	-6.08	-1.50	0.50	-1.52
Error reduction	5.71	-22.49	-8.82	2.38	-5.80

Profile Type: Custom; Application: Integers; Gender: Female

The integer tests used strings of integers ranging from two to five integers in length. The word 'and' was also permitted. There were 100 test utterances for each speaker. The grammar permitted an infinite string of integers in any combination.

Summary of Results:

Overall Results: Profiled vs. Non-profiled Tested with the Single-Transcription Dictionary

The table indicates the performance increase and percentage of error reduced for each speaker for the single-transcription (STD) model over the generic model (3008).

Table I. Improvement: Single-Transcription Profile vs. Baseline Both Tested with Single-Transcription Dictionary

Speaker	carla	gloria	jacque	pam	MEAN
Word Accuracy (%)					
Recog. increase	6.28	5.90	14.76	2.58	7.38
Error reduction	27.01	28.06	47.05	13.98	29.02
Utt Accuracy (%)					
Recog. increase	14.00	6.00	23.00	6.00	14.75
Error reduction	25.45	31.37	37.10	15.00	27.23

Profile Type: Custom; Application: Integers; Gender: Male

This section presents the results of testing integers for the males. The same conditions apply to the males as for the females.

IV. Summary of Results

Overall Results: Profiled vs. Non-profiled Using Single-transcription Dictionary

Table I. Improvement: Single-transcription Profiles Over Baseline

Speaker	bob	bowen	bryan	james	MEAN
Word Accuracy (%)					
Recog. increase	2.95	8.37	4.06	3.32	4.67
Error reduction	24.98	46.68	34.38	21.94	31.99
Utt Accuracy (%)					
Recog. increase	7.00	17.20	7.00	2.00	8.30
Error reduction	21.88	42.09	25.93	5.88	23.95

Results for Gang Profiles

The gang profiles pool the data by gender. The intent of this procedure is to increase performance without the need to be constantly switching models. It is also useful in enhancing the generic model for other speakers using the same application texts.

For each group, three models were built using these concatenated lists of utterances: one gang-profile, one gang pcb augmentation of weight 1, and one gang pcb augmentation of weight 3 (the files of label populations were given three times the weight before contributing to the codebook).

Each of the eight speakers contributed 375 test utterances independent of the learning set (the same as was used to test the custom models). As noted above 60 test utterances from one of the male speakers (bowen) were not usable (these were primarily NASA command utterances) so he only contributed 315. All scores below are averaged across the speakers, so female model evaluations really involve 1500 utterances (4 x 375), and male model evaluations involve 1440 utterances, since 60 are missing.

The text for profiling was the same as for the custom models. The gang models were tested on Sun SPARC Stations (faster than the Sun 4/110 used for custom testing) and an improved decoder version was used. Therefore the time results are not directly comparable with the custom profiles. Otherwise the same analysis conditions apply such as slider setting 6 and language weight 0.

The tables are labelled as follows nasaonly: NASA commands, and nasatst: integers.

Profile Type: Gang; Application: Both; Gender: Female

The female model used the following amounts of data to perform the profiling: carla 1129, jacque 1143, gloria 1148, pam 1150. The total was 4570 utterances.

Table x. Gang Profile Results for Female Speakers

baseline 3008	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
nasaonly	93.43	94.55	79.62	89.12	95.25	42.39	1.78
nasatst	89.23	89.58	68.08	82.67	93.75	36.36	2.42
gang-profile	**Word Acc**	**W Trans**	**Utt Acc**	**+ Off 1**	**+ Off 2**	**Segs/Utt**	**Ave Time**
nasaonly	96.45	96.61	84.00	92.88	98.50	40.16	2.51
nasatst	93.40	93.59	76.67	90.50	98.00	35.01	3.25
augment*1	**Word Acc**	**W Trans**	**Utt Acc**	**+ Off 1**	**+ Off 2**	**Segs/Utt**	**Ave Time**
nasaonly	94.85	95.74	81.88	90.62	96.62	42.56	1.66
nasatst	91.73	91.82	72.83	85.83	96.00	36.48	2.28
augment*3	**Word Acc**	**W Trans**	**Utt Acc**	**+ Off 1**	**+ Off 2**	**Segs/Utt**	**Ave Time**
nasaonly	95.39	96.21	83.75	91.38	97.12	42.56	1.56
nasatst	92.78	93.09	76.00	88.00	96.83	36.48	2.18

Profile Type: Gang; Application: Both; Gender: Male

The following amounts of data were contributed by the speakers to the gang profile: james 867, bob 1149, bowen 923, bryan 1150. The total consisted of 4089 utterances. The number of utterances for james is less than the number used in the custom model due to an initial difficulty in file transmission. This was subsequently rectified and the custom model rebuilt, but with little improvement in the profile.

Table x. Gang Profile Results for Males

baseline 3009	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
nasaonly	92.71	95.04	79.14	89.57	95.19	42.28	2.17
nasatst	90.90	92.37	74.76	87.55	95.44	36.40	2.86
gang-profile	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
nasaonly	94.16	95.54	80.61	90.37	95.72	41.09	3.16
nasatst	91.95	93.19	77.48	89.40	95.53	36.40	3.78
augment*1	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
nasaonly	93.11	95.34	80.61	90.64	95.32	42.28	2.14
nasatst	91.64	93.13	77.13	88.96	95.79	36.40	2.67
augment*3	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
nasaonly	93.87	95.88	82.62	91.58	95.86	42.28	1.90
nasatst	92.54	93.86	79.32	90.27	96.41	36.40	2.45

Gang Profile Summary

For both females and males, these generalizations can be made:

1) All three models show improvement over baseline performance.

2) Augmenting with three times (*3) the weight is better than augmenting with regular weight (*1).

3) Gang-profiling shows higher accuracies than augmenting with regular (*1) weight, although gang-profiling is slower.

Now to be specific about male and female performance (performance now being narrowed down to comparing the gang-profiled model with the augmentation done at three times the weight):

For the females, the gang-profiled model gets slightly higher word and utterance accuracies (~1%) on the 'nasaonly' and 'nasatst' tests. The pcb augmented model gets exactly 1% higher on word accuracy on the 'nasa_phobal' test.

For the males, the gang-profiled model gets higher WORD accuracy on 'nasaonly' and the pcb augmented model gets higher UTT accuracy; the gang-profiled model gets nearly 3% higher word accuracy on the 'nasa_phobal' test; the augmented model gets higher word and utt accuracy on 'nasatst'.

Adaptation Summary

Eight speakers of American English representing a dialect continuum from GA to Southern (Southwestern) were tested on a total of 375 utterances each using three models per speaker and two dictionaries.

The profiled models performed better than the baseline generic models. Little or no difference could be found between the two dictionaries across all speakers and texts.

Recognition improvement is very difficult to rank. Custom profiles, gang profiles, and gang pcb augmentation (no custom pcb augmentation was performed) all showed very similar improvements. Dictionary transcription changes clearly showed the least improvement.

For the females on the NASA commands task the precise figures are:

```
Mean Custom Profile Improvement (word accuracy):  2.88%
Gang Profile Improvement:                         3.02%
Gang PCB Augment * 3:                             1.96%
Multiple-Transcription Dictionary:                0.75%
```

Note that for the females the gang profile provides more improvement than custom profiling. However, if one averages males and females then custom profiling has a very small advantage. This does highlight the variation and difficulty of predicting performance increases *a priori*.

The mean word accuracy improvement after profiling was, across all speakers and conditions, approximately 5% for the single-transcription dictionary (STD) profile. This improvement resulted in a mean error reduction of 23.71%, nearly a quarter of the errors eliminated.

2.3.5. Discussion

The results are discussed by profile type, application and gender.

Profile Type: Custom; Application: NASA Commands; Gender: Female

The maximum utt accuracy is 88.5% for gloria profiled with the single-transcription dictionary and tested on the multiple transcription dictionary. The maximum increase in utt accuracy is 8.5%, again for gloria profiled with the single-transcription dictionary, but tested on the multiple-transcription dictionary versus the baseline. This is a 42.5% error reduction. All other speakers achieved their best performance (utt accuracy) using the single-transcription (standard dictionary) profiles with the single-transcription dictionary. This result would suggest that a multiple-transcription dictionary can be helpful for recognition of some female speakers, but probably not for profiling them.

Profile Type: Custom; Application: NASA Commands; Gender: Male

The highest utt recognition score was achieved for bob (83.50%) using the model profiled on the standard dictionary (single-transcription 'the') and tested on the same dictionary. The maximum improvement (utt accuracy) was for bowen using the model profiled on the multiple-transcription dictionary and tested on the same dictionary (8.11%; 23.08% error reduction). Two speakers (bob and james) were best recognized with the single-transcription model and dictionary (utt accuracy). Two speakers (bowen and bryan) were best recognized with the multiple-transcription model and dictionary. Bryan's improvement over baseline was 2.5% with 12.82% error reduction. The speaker with the dialect most divergent from General American appeared to be bob. The profiling was very effective for him. The reasons for the differences in performance are not clear, but they are fairly small differences anyway. One possible reason for recognition differences, rate of speaking, was checked with a rather rough measurement, but rate appeared to be within normal limits for all speakers.

Although two speakers were recognized best with a multiple-transcription setup (utt accuracy; three speakers for word accuracy), the improvement was modest compared to the single-transcription setup (see male Table VI, above).

These figures when considered along with the female results still do not appear to justify special dialect-specific dictionaries for NASA speakers.

Remaining questions to be investigated are reasons for relatively poor performance by some speakers, and tests of digit and phobal recognition. Also the reason why males tend to have a slight preference for the multiple-transcription dictionary should be investigated. (It is possible that males are more divergent in their pronunciation, which is a well-known sociolinguistic characteristic of American English.)

Profile Type: Custom; Application: Integers; Gender: Female

The maximum word accuracy achieved was for gloria (85.98%; 3008_mult model with single-transcription dictionary STD). The maximum utt accuracy achieved was for pam (68%; 3008_mult model with multiple-transcription dictionary MTD). The maximum word accuracy improvement was for jacque (14.76% over baseline; 47.05% error reduction; 3008_single model with STD).

The best results for each speaker were achieved using different combinations of profiled models and dictionaries. The best combination for carla was 3008_single model with MTD. The best combination for gloria was 3008_mult with STD. The best combination for jacque was 3008_single with STD, and the best combination for pam was 3008_mult with MTD.

The 3008_mult models with the MTD gave the best overall performance (mean = 84.22). The 3008_single models with the STD were only slightly inferior (mean = 83.86). This slight difference does not appear to justify using multiple-transcription dictionaries in general. However, certain integers might benefit from a multiple-transcription. This possibility will be studied further.

Profile Type: Custom; Application: Integers; Gender: Male

The maximum word accuracy score was 93.36% (bryan, 3009_mult model with MTD), and the maximum utt accuracy score was 83.00% (bryan, 3009_mult model with the STD). The best over-all models were 3009_singles (mean word accuracy= 90.50%; mean utt accuracy = 74.84%).

The best individual word accuracy scores were 92.25% for bob, 91.24% for bowen, 93.36% for bryan and 88.19% for james. Three of the speakers achieved their best results using 3009_mult models. Three of the speakers achieved their best results using the single-transcription dictionary (STD). The mismatch may be explained by the fact that a few single-transcription entries are less optimal than multiple-transcription entries. Also, the multiple-transcription labels may more accurately capture certain acoustic events while the single-transcriptions may limit the recognition possibilities more rigorously.

The mean improvement from the baseline using the 3009_single models with the STD was 4.67% better word accuracy and 31.99% error reduction. Once again the results favor using the simpler single-transcription dictionary with possible improvements to certain transcriptions yet to be specified.

Note on Gang Profiles

For females the gang profiles gave more improvement than pcb augmentation while for males the reverse was true by a small margin.

2.3.6. Conclusions

The study indicates that speaker profiling is preferable to generating a new dictionary for dialect speakers. Profiling resulted in approximately a 5% improvement in utterance accuracy depending upon the grammar and text used. For the females a comparison of results using the generic models found only a 0.75% improvement for the NASA_dialect (MTD) dictionary over the standard dictionary for NASA commands (males were not tested with both dictionaries on the generic model). The NASA_dialect dictionary did not meaningfully improve recognition for any of the profiled models either (all of which were tested on both dictionaries).

It could also be noted that the male and female speakers with the strongest dialects improved the most with profiling. Since there were only a few speakers in the study one must be rather tentative about that conclusion.

While the question of which profiling type shows the most improvement is somewhat dependent upon sex and task, overall (in this study) the custom profiles gave the most improvement, followed by gang profiles and weighted pcb augmentation. The last two were very close overall.

Another advantage of profiling is that the models are trained on the vocabulary of the application (apart from any considerations of dialect). Thus profiling should show significant improvement in words which were not well represented in the original model construction, and were somewhat confusible.

Profiling does not usually provide much improvement in cases where the speaker performs close to the level of dependent speakers (speakers used to train the original model), and where the application text has no marked differences from model building text. The latter criterion is difficult to quantify, and will vary from model to model as the texts used to build the model vary.

Some areas in which additional research would be useful are the effect of profiling upon speakers who differ from the mean by variables such as rate of speaking and to attempt to measure textual characteristics so that one can determine whether the application is sufficiently different from the model building material to warrant profiling.

NASA Delivered Models

The dialect development subtask of the project has produced many models adapted for NASA speakers and the Southern dialect (as represented by the NASA speakers). The actual models delivered to NASA were the best of the ones built in this subtask. Specifically, the custom delivered models are the custom single transcription dictionary profiles, one for each speaker for a total of 8. The generic delivered models are the gang profile female model and the gang augmented (*3) male model. These provided the best improvements on NASA applications as evidenced by the test results above.

2.4. HEADSET DEVELOPMENT

2.4.1. Introduction

One of the major selling points of speech recognition systems is that they allow for interaction with computer systems when a user's hands are busy. Many prospective customers have told us that they require headset capability for their "hands-busy" applications. Originally, SSI delivered systems with a handset and cradle as the front end to the phonetic engine (PE). For this subtask of the project, a headset capability has been developed both for NASA's use and for the enhancement of our product in general.

The original handset and cradle input device to the phonetic engine looked like and was operated by the user much like a telephone. This made it easier for casual users who had never dealt with a headset to use our system. However, we have since determined that the majority of the speech recognition market does not find a handset to be the input device of choice. This is because the users of most current applications of speech input are already familiar with headset usage. A headset is preferred for extended use of speech input because the hand position for a handset can become tiring, as well as for other ergonomic reasons.

SSI made the decision to change completely from a handset input device to a headset input device, based largely on these considerations. This change took place with our release 3.0, the PE200 release. Earlier releases, using the PE100, have all had handset speech models; releases since 3.0, using the PE200, have all had headset speech models. This was somewhat different from our initial plan for this project, which was going to support both handset and headset speech models. Perhaps as speech input becomes more common with better performance, the emergence of casual users will require a handset again. Some "over-the-phone" applications still use telephone handsets as the input device.

The different acoustic characteristics of a headset microphone meant that it could not simply be substituted for the handset. Different speaker models had to be created for the new input device. This document describes the work that was performed for adding a headset capability to SSI's speech recognition system.

We start by describing our requirements for the headset and how the chosen headset meets those requirements. We report the results of initial comparison tests between the handset and the headset. Several problems we encountered during the conversion are described along with their corrections. Then we give more reliable comparisons between the headset and handset, by comparing the different releases using the different input devices. Finally, we include a summary of our work on the project and draw some conclusions from the results.

2.4.2. Headset Requirements

There are two sets of requirements for any speech input device: ergonomics and acoustic/electrical performance. Of primary importance are the acoustic/electrical requirements. After all, if it does not work well no one is going to use it even if it is the most comfortable unit in the world.

(A) Acoustic/Electrical Requirements:

1) "Good" frequency response. This means it should be relatively flat from about 20 Hz to about 10 or 15 kHz with a "graceful" role-off at each end. This response should also be consistent from unit to unit. The low frequency response requirement eliminates dynamic microphones and leaves us with electret condenser microphones.

2) Should be an electret condenser microphone. Although 1) indirectly makes this a requirement we required it independent of 1) because the microphone in our original handset was also an electret unit and we wanted to retain as much compatibility between the two as possible.

3) The unit should be a directional microphone (in microphone industry terminology, a "noise canceling" microphone). This allows us better rejection of background noise.

4) The unit should have good noise and distortion properties. Almost any reasonable unit from a reputable manufacturer will behave similarly in this area.

(B) Ergonomic Requirements:

It was decided that a proper headset for our applications should be as unobtrusive to the user as possible while still meeting the acoustic/electrical performance requirements.

2.4.3. Chosen Headset

This chosen headset is the AKG model Q15/410. It is a special model from the manufacturer, essentially combining our acoustic/electric and ergonomic preferences from the input devices considered in our headset investigation.

The AKG unit meets the acoustic/electric requirements and is reasonably priced (though it is not cheap). Extensive electrical/acoustic testing at SSI has verified the performance of many units as well as the consistency between many units.

The AKG unit does not cover the ears and is a small, light unit. Wearing it is very similar to wearing eyeglasses, but it does not get in the way if you are wearing eyeglasses also. As long as your ears don't object to the eyeglass-type mounting, the unit meets the ergonomic requirements.

2.4.4. Initial Tests

The initial tests were for custom speaker models. They compare newly built headset models with previously built handset models. Results are given here for two speakers' data, one male and one female. Comparison of results of headset vs. handset performance for generic models will be described later.

The initial models used an initial version of autolabelling, a technique for automatically labelling speech data by using a previously existing speech model. The new headset speech data was generally autolabelled with the speaker's corresponding (custom or generic) handset model. While not necessarily giving labellings as good as expert labelling, the automatic labellings are good enough for model building. Furthermore, they may be more appropriate for improved recognition because they reflect label assignments chosen by the recognizer itself. Autolabelling has been evaluated independently of the headset/handset issue, and is part of many of SSI's model building processes (in particular, profiling).

Here are the results on two of the first custom headset models, for one male and one female speaker. These tests use the "atb4" test grammar, a common test grammar in use during the time of these model builds. These tests are on about 250 utterances per speaker.

```
speaker:        RKR (male)

test                 % utts      % words    % tran      CPU sec/utt
----                 ------      -------    -------     -----------
custom handset       47.54%      70.35%      68.48%       23.40
custom headset       60.66%      79.42%      77.10%       27.19

speaker:        AM (female)

test                 % utts      % words    % tran      CPU sec/utt
----                 ------      -------    -------     -----------
custom handset       25.17%      54.81%      54.07%       17.40
custom headset       38.84%      67.40%      65.98%       31.60
```

These results show a large accuracy improvement from changing the input device. This is probably due to some known acoustic problems with the handset (specifically, poorly understood acoustic properties of the microphone arrangement in the handset and inconsistent placement of the handset when in use). We could not have reasonably expected better results than this. The initial results indicate that we can change to our selected headset, meeting a customer request/requirement, and achieve an accuracy improvement with the same change.

There was some concern about the increase in cpu time in these initial results. But this was considered to be less of a concern, because of other reductions in decoding time that were rapidly becoming available. It turned out that the time degradation of the second test model above was an isolated artifact of that model.

2.4.5. Initial Problems

Next came the process of fully integrating the new input device into our standard system. During this process of productization, we encountered several problems with recognition, where the new results were not as expected from our initial tests. These were generally solved after some investigation by being very specific about what exactly is our input device, i.e., by making sure all headset configurations are as specified. In particular, the AKG Q15/410 is used, with its windscreen, and with all new PE200 acoustic processing.

Several tests were run to decide on this exact configuration of the input device. We give the results of some of these tests here.

(A) Windscreen vs. No Windscreen

In some of our initial tests we were rather cavalier about whether or not the windscreen was used. When recognition degradations were noticed if the windscreen was not used, a test was initiated to evaluate the effects of windscreen usage on recognition. One such result is given here. This test again uses the atb4 test grammar as in the earlier tests.

speaker: AM (female)

test	% utts	% words	% tran	CPU sec/utt
without windscreen	38.84%	67.40%	65.98%	31.60
with windscreen	55.60%	77.48%	75.39%	31.18

Recognition clearly improves when using the windscreen as compared to not using the windscreen. This was found to be true with other speakers and generic speaker models as well. Subsequent tests and our default system configuration all now use the windscreen.

(B) PE100 vs. PE200 Acoustic Processing

Essentially, all of our headset models have been built to work with the PE200 version of the Phonetic Engine, while all handset models have been built to work with the PE100 version. There are several differences between the PE100 and the PE200 other than the input device that affect recognition. These changes were implemented to improve recognition. One such change is the normalization of the filterband gain values, where the PE200 supports a finer resolution of these values (1024 instead of 200). Several tests were made to make sure that this and other changes to the PE were beneficial to recognition. The results of a digits test on 400 utterances for a single speaker are shown here. These are both headset with windscreen tests.

speaker: LGW (male)

test	% utts	% words	% tran	CPU sec/utt
PE100 normalization	85.75%	97.71%	96.57%	0.74
PE200 normalization	86.00%	97.77%	96.40%	0.84

There is not much difference in these results. The PE200 version is expected to provide more benefit for generic models involving more data from more speakers. The test here shows that the change to the PE200 version has not degraded performance of custom headset models, and has arguably improved it. (A test of the generic headset model was not performed, because it would require data from several independent speakers using both the PE100 and the PE200, a major data collection task.)

(C) "Green" vs. "Black" Headsets

The manufacturer of our chosen headset makes two headsets that look almost identical except for a small spacing bar on the microphone boom. The higher quality acoustic/electrical headset (our headset choice) has a green spacer while the other has a black spacer. Somehow we received several of the "black" headsets by mistake. Discovering this difference between the "black" and "green" headsets was a major part of the hardware debugging effort of the PE200 headset capability and of the PE200 hardware itself.

After construction of some of the first PE200s, many were found to have much lower performance than was expected from off-line tests. We first thought that there was a problem in the acoustic processor. Individual parts of the acoustic processor were tested until the problem was traced back to the microphone itself. Independent microphone tests for consistency had discovered the difference between "black" and "green" headsets in terms of acoustic/electrical response. The difference in recognition performance between the two headsets was much greater than expected.

Here are the average results of several test runs of one speaker testing groups of 70 utterances from the pmtext application (another test application). Several test runs were made to eliminate other variables in the test.

```
speaker:          DJM (female)

                % words correct
 test      max-min     std.dev.      mean
-----      -------     --------      -----
black      17.82         6.41        58.36
green       6.57         2.40        81.56
```

These results show that recognition is improved when using the correct headset. The variation among repeated tests is also smaller when using the correct headset. The "green" headset has also been shown in other tests to be better for both dependent and independent speakers on generic models.

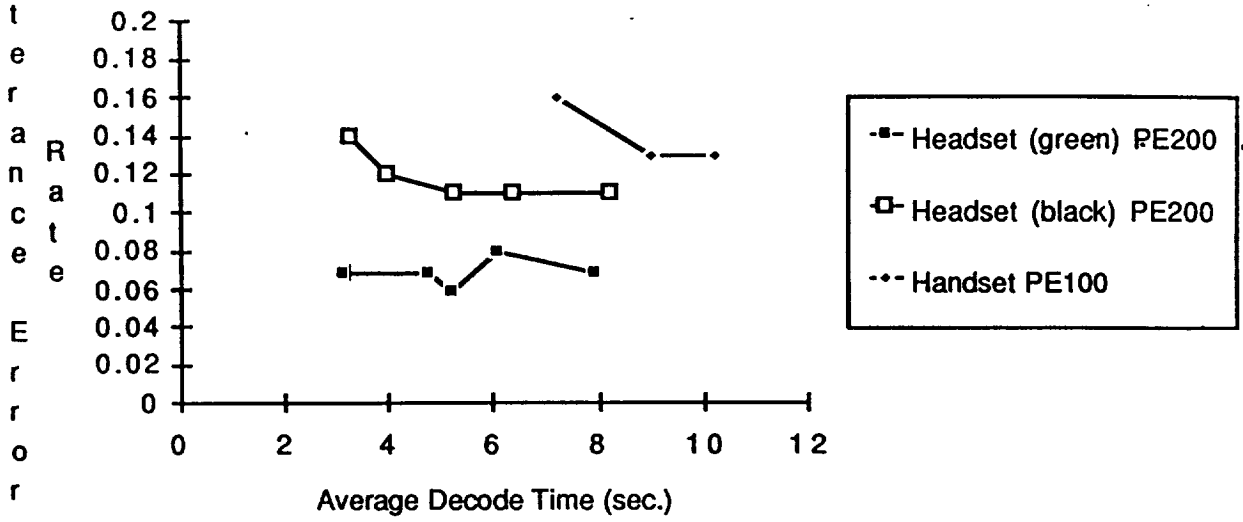
Unfortunately, this test also shows that our speaker models are still quite sensitive to the particular microphone in use. Other subtasks of this project will address this issue somewhat, particularly the task to reduce noise sensitivity.

2.4.6. Headset vs. Handset Release Performance Comparisons

Several tests were made comparing headset and handset performance as part of the tests comparing the PE100 with the PE200 for release 3.0. These tests show that we have achieved our goal of adding a headset capability with performance at least as good as that of our handset. Actually there is quite an improvement in the release using the headset. But there are many other differences between release 3.0 and the previous release (2.3). The overall performance differences will be seen from the results presented in this section.

A summary figure of several tests comparing release 2.3 (PE100 handset) with release 3.0 (PE200 headset) is given in figure 2.4.1. These tests are all with generic female models, on a digits recognition application. The tests using the headset release are generally faster and more accurate. Results with the "black" incorrect headset are also included for comparison. Its performance lies between the curves of the two actual releases.

Figure 2.4.1. PE100 Handset and PE200 Headset Comparison



From figure 2.4.1, we can see that the utterance error rate for the headset release is approximately half of what it was in the previous handset release. Also, the slowest setting of the newer release is only slightly slower than the fastest setting of the older release. Roughly, the time required to decode the average utterance is about half of what it was under similar time-accuracy tradeoff circumstances.

2.4.7. Summary and Conclusions

To summarize our work in developing a headset capability for SSI's speech recognition system, we have: specified headset requirements, selected a headset meeting those requirements, compared the chosen headset's performance with our earlier standard handset, and integrated the chosen headset into our standard delivered speech recognition system.

The headset system meets a need both for NASA's use and for our other customers. These and other such marketing considerations have encouraged SSI to use a headset as the speech recognition system's standard input device.

We have selected a headset that gives performance better than our original handset, so there is no loss in system performance for changing to use the headset. Using the headset is also less expensive than having to build our own special handset, as was previously required.

Release 3.0, which converted to using the headset as the default, provides further speed and accuracy system performance improvements, as have all subsequent releases. The headset project has been a complete success.

3. GENERAL PERFORMANCE IMPROVEMENT

3.1. PARALLEL PROCESSING IN THE PHONETIC DECODER

INTRODUCTION

One barrier to market acceptance of automatic speech recognition has been system response time. Our Phase I results indicated that while the system recognizes speech at a practical level of accuracy and tolerance for noise and stress, its utility would be enhanced by a speedup. In addition, a speedup in processing would allow more processing time to reduce noise effects.

The main time bottleneck of the system is in the Phonetic Decoder. The Phonetic Engine (PE) works faster than real time regardless of application complexity, and already uses parallelism (the three microprocessors in the PE run in parallel). The Decoder, on the other hand, is affected by application complexity (e.g., the branching factor of the syntax, the phonetic confusability of the vocabulary) and works on a principle of a speed-accuracy tradeoff: the more time available, the higher, generally, is the recognition accuracy. It must search through the set of acceptable syntactic paths and word transcriptions. The more paths it searches, the more likely it is that the correct one will be found.

With large vocabulary continuous speech, however, there are an enormous number of possible paths, far too many to perform an exhaustive search and still maintain a "reasonable" response time. Our solution has been to adopt a heuristic search strategy, where only paths which appear likely are searched. The less likely paths, based on the evidence so far, are not given further consideration. If a partial path at first appears unlikely, even though later processing might make it the most likely path globally, it may be discarded. This is an unfortunate fact of life in a heuristically driven left-to-right search strategy, which is a part of virtually all current speech recognition systems. If, however, many paths could be searched in parallel, real-time constraints would be lessened and more paths could be considered. This would increase the probability that the globally optimal solution is found. By adopting parallelism in the Decoder, we expect that the ratio of accuracy to decoding time would increase.

We have taken a conservative approach to considering parallel processing for the Phonetic Decoder, by modifying our original sequentially oriented version to be more suitable for processing in parallel. The bulk of the actual modifications made to the Decoder are in this category of "Decoder Re-Engineering for Parallelism." These modifications consist of preparations for allowing the Decoder to perform its work in parallel. These changes alone have produced major improvements to the system in both decoding time and accuracy. The changes made in this area include parallel word scoring, dynamic beam sizes, relative thresholding, coarticulation simplifications, unique word scoring, and parallel path scoring via a single-level decoder. This final report includes the results of all of these studies in the section on "Decoder Re-Engineering for Parallelism."

With the Decoder prepared for parallelism, estimates have been made of the speedup available from actually porting the Decoder to a parallel processor/system. We have assumed as a target parallel processor a communicating network of processors, similar to our SUN network. This model is used to determine when a parallel implementation becomes worthwhile, depending on the communication overhead time. We have also

measured the communication overhead typical of our SUN network and drawn conclusions about recasting the Decoder to work over our network. This report includes all of these predictive studies in the section on "Parallel Processing Predictions."

Finally, we consider some related work and include a summary of our work on this project and the conclusions we have drawn from our results.

3.1.1. DECODER RE-ENGINEERING FOR PARALLELISM

Prior to the start of this project, the Phonetic Decoder used a two level beam search. One level of the beam search was used to score individual words, and the other level was used to put words together into partial paths, given syntactic considerations. The Decoder's basic algorithm then could be specified using a pseudo-code description such as the following:

```
FOR EACH speech segment 'X'
  FOR EACH word 'W' that can begin at segment 'X'
      BEGIN
        score word 'W';
        record results;
      END;
```

Each word is scored separately in this case. A parallel implementation could simply do all of the word scorings as separate processes. This assumes that there are a sufficient number of processors to score all the words in parallel. Since there are usually many words that start at a given segment, there will usually not be enough processors for every word. One way around this problem is to group words together into packets to be scored together, in an attempt to keep all available processors as busy as possible.

This apparently simple step of re-engineering the decoder to group words into packets for parallel scoring has had many repercussions, leading to a better understanding of where competition between paths is important for efficiency and accuracy. Our first step was to prepare word packets for scoring in parallel. (The terminology is somewhat confusing here since the words in one packet are sometimes considered to be scored "in parallel," when we really mean that the words in one packet are scored together, or simultaneously. Only the individual packet scorings would be executed in parallel in a parallel implementation.) Other reorganizations were made to the Decoder to make its word scoring level and its syntactic path keeping level more independent. This independence was accomplished through the dynamic beam size and relative thresholding sub-projects.

During this process, several other ideas for improving Decoder efficiency were implemented. A primary reduction in the computing resources used was achieved by scoring a given word only once at the start of a given segment. This is what unique word scoring implements. It is a reduction in the computing resources required for either a sequential or a parallel implementation. Coarticulation simplifications were necessary to implement this, because in earlier versions different instances of the same word starting at the same segment might have different coarticulation contexts.

Analysis of the results of earlier steps revealed that the competition among words within one packet of words was having a beneficial affect on both decoding time and accuracy. Thus to increase this effect, we produced a version that scores not only all words simultaneously, but also all paths simultaneously. This is implemented by combining the

work of the two levels of searching into one level, resulting in a one level beam search algorithm. This algorithm's pseudo-code is given here:

```
FOR EACH speech segment 'X'
  FOR EACH partial path 'P' active at segment 'X'
      BEGIN
        score next possible phones of partial path 'P';
        record results;
      END;
```

The parallel implementation of this final version is much different from the one we planned to move toward at the start of this project. Parallel processes would be used for scoring (groups of) parts of words rather than whole words or whole word groups. The general plan of making the inner, time expensive scoring portions of the algorithm run in parallel remains valid. However, this version would be designed to allow the innermost phone (instead of word) scoring portions to run in parallel. (A description of an implementation of this final algorithm in parallel is given in part 3.1.2.)

We conclude part 3.1.1 with a summary of the performance changes resulting from each step of the project.

3.1.1.1. THE PARALLEL WORD SCORING PROJECT

Introduction

"Parallel Word Scoring" is a Decoder speed enhancement project that prepares groups of words to be scored in parallel (i.e., the groups are scored in parallel; the words within a group are scored simultaneously). This is in preparation for some eventual port of the decoder to a parallel processor; scoring of different word groups can take place in parallel. For this project though, the words in a group are considered to be scored in parallel, or together, rather than sequentially. This is to increase the competition among words by having them compete at an earlier stage in their scoring.

Prior to this project the scoring algorithm had words competing only after they had been completely scored. Parallel word scoring introduces inter-word competition to the segment beam level of the decoding algorithm, by having a group of words share the segment beam space resource.

The impact of parallel word scoring on accuracy was a major concern. The plan, however, was to implement the project fully parameterized, allowing the user to set the number of words to score in parallel, and to allow a larger beamwidth for the segment beam. By setting these two parameters accordingly, parallel word scoring would maintain the current system's speed/accuracy capabilities.

Having the capability to score words in parallel makes the decoding algorithm more versatile and more adaptable to future projects, such as pipelining, parallel architectures, and avoiding word scoring duplication.

I. PARALLEL WORD SCORING: PLANS

By scoring words in parallel, and assuming the same search configurations, (e.g beam widths), we increase the competition among words by effectively decreasing the available resources. Whereas previously a word was scored utilizing the entire search space, with parallel scoring, the word must share, and compete for, that space with several other words. This increase in competition will result in a great time savings.

Decreasing search resources would have a substantial negative impact on accuracy. Obviously if reducing the search space was our only goal we could have simply done so without implementing parallelism. What we envision happening is that the better scoring words will eliminate immediately the "garbage" words, and then the search space will be dedicated only to those more probable words. Nevertheless, we do expect an initial accuracy loss with the first implementation, but only minor. By "re-tuning" the search parameters with respect to the time savings incurred by parallelism, we expect ultimately to yield an increase in accuracy (yet maintaining the speed improvements).

By increasing inter-word competition, parallel word scoring strengthens the impact of a beam search by strongly biasing utterance transcriptions with higher scoring initial portions. However, it may harm transcriptions with poorly scoring initial portions. These scoring progression types are compared in the following Figure 3.1.1.1.1.

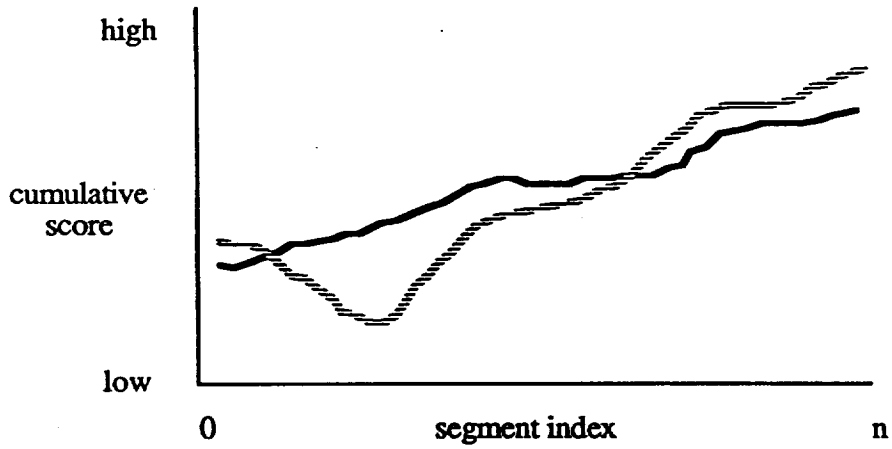


Figure 3.1.1.1.1. Transcriptions with different score progressions.

Transcriptions that take the path of the broken line are most apt to feel the effects of parallel word scoring, while those of the solid line will benefit from it, in terms of decoding time. The reason for this is that intermediate word scores will be in competition with other words, whereas currently only word-final scores compete.

The configuration of the decoding algorithm prior to parallel word scoring allocates the entire segment beam to each word being scored; 1 word gets a beamwidth of N. Scoring W words in parallel will put this ratio at W:N (N being the segment beamwidth normally used for one word, and W the number of words being scored in parallel).

Potential accuracy problems with parallel word scoring can be eliminated by increasing the segment beamwidth adequately. If W words are scored in parallel, then the beamwidth can be set to W*N, putting the ratio of words to beamwidth back to 1:N. However, it must be noted that setting the beamwidth so high will not insure the same accuracy. Setting the

beamwidth to W*N only insures the same resource ratio; it may be that many good scoring words will take up more than their fair share of the total beamwidth. A word-to-beamwidth ratio of W:N would be the (predictably) fastest/least accurate setting, and a ratio of W:N*W would be the slowest/most accurate. Parallel word scoring enhances the speed/accuracy curve by allowing for word-to-beamwidth ratios that are currently unobtainable. Scoring words in parallel allows for the ratio to fall between W:N and W:W*N. A number 'r' (where $0 < r <= 1$) can be chosen such that the ratio

W: N * W * r

will yield the most efficient speed/accuracy tradeoff. As r approaches 1 the system slows and becomes more accurate, as r decreases to 0 the decoding speeds up and accuracy drops. As a fully parameterized function, parallel word scoring has a great potential for speeding the decoding time with a minimum effect on accuracy. The speed increase will vary with 'r'.

More On Accuracy

The extra competition introduced by parallel word scoring can be minimized by restricting the word parallelism to words within a syntactic category. All the words in a word category belong to the same parse, thus the extra competition will remain within a single parse. Since we only retain one instance of a parse ending at any given segment, this speed increase gives no accuracy decrease.

More On Speed

Estimating the actual speedup of parallel word scoring is cumbersome due to the "snow-balling" effect that it will have throughout the decoding process. Parallel word scoring will examine fewer transcriptions per word, which will mean less work spent on the word beam and segment beam. And as fewer words are placed on the partial path beam, so in turn fewer calls will be made to the segment beam for words to be scored.

Implementation

From the command line, the user, or application, will be able to control the parallel word scoring function, just as if it were a threshold or a beamwidth. It may be advisable to include parallel word scoring as another aspect of the slider settings, thereby keeping this transparent to the user. (Slider settings are combinations of decoder search parameters (beamwidths, thresholds) which have been preselected for good time-accuracy tradeoffs.)

The new controllable parameters will be:

- Only score word categories in parallel.
- Number of words (W) to score in parallel (size of word packet).
- Segment beamwidth (N) per word.
- Segment beamwidth adjustment (r).

the actual segment beamwidth will become W*N*r.

By allowing the user/application to set the number of words to score in parallel the system can maintain the current decoding behavior. When the number of words to score in parallel is set to ONE, the system would be reduced to the current configuration.

Parallel Word Scoring and Pipelining

"Pipelining" is the project that has the PE and PD (and speaker) working in parallel. The PD will be sent phonetic codes in packets and will begin processing before all codes have arrived from the PE. A major design issue for the PD in pipelining is how to handle a "wait" or "blocked" condition, which would arise when the PD needs more codes from the PE. Parallel word scoring can simplify the "wait" state problem by having many (or all if possible) sentence initial words scored in parallel, thereby reducing the likelihood of a "wait" state to ever occur.

If pipelining were the only objective (and not decoding time) the decoding search could be organized so that each word would still have its own beam, but they are all scored in parallel; this would avoid inter-word competition.

Conclusions

Parallel word scoring will shorten the decoding time by increasing the scoring competition. However, its time savings may be at the price of accuracy, but not in all situations. Parallel word scoring is a true enhancement to the system in that it maintains the current system's capabilities of speed/accuracy (it can be set to score one word at a time) while adding a new potential for the speed/accuracy tradeoff curve.

II. PARALLEL WORD SCORING: RESULTS

The new version of the Decoder decodes utterances by scoring words in "parallel." That is, the decoder now scores more than one word at a time, by forming groups of words to be scored together. The actual number of words decoded in parallel is defined by the slider settings.

To achieve the maximum efficiency from parallel decoding it was necessary to reset the slider setting values. As it turned out the speed/accuracy improvements seen as a result of the parallel decoding implementation were more attributable to the new slider settings than to parallel word scoring. Still, parallel word scoring was valuable as a predecessor to other upcoming projects.

Another new option offered to applications is to restrict parallelism to word categories. That is, only words belonging to the same word category may be scored in parallel. The behavior of this option has basically gone untested.

Many changes were made to the Decoder Kernel code as a result of implementing parallel word scoring. The most significant changes are centered around the segment beam. When the parallel word scoring version is made comparable to the previous release of the Decoder by scoring only one word in parallel, the new version is about 1% slower. This is due to the overhead associated with maintaining parallel decoding. This overhead is apparently minimal.

The communication interface between the two levels of decoding (the path beam and segment beam) has also been changed. Now the path beam groups words together and sends the segment beam packets of words for decoding. In previous versions the word beam simply requested one word at a time for decoding, so that no major interface was required.

Conclusion

The primary difference to be seen in the new version is the approximately 10-30% speed up, from parallel word scoring and the subsequent parameter setting / slider setting task.

3.1.1.2. DYNAMIC BEAM SIZES: DESCRIPTION AND RESULTS

Introduction

This project has made the beams of the decoding beam search more versatile. The path and segment beams of the Phonetic Decoder have been modified to be dynamic to a limited degree. This helps to clean up several rough edges of the decoding algorithm. Beamwidth limitations no longer produce randomness due to scoring ties. Search at the beginning of an utterance is not pruned as severely as later on in the utterance. These changes make the two levels more independent of each other, in preparation for any upcoming implementation to have them execute in parallel.

Segment Beam Changes

The segment beam's size will now expand dynamically to keep any transcription path element that has the same score as the worst scoring element in the fully pruned beam. Thus ties are no longer decided randomly; instead, a score must be lower than the minimum saved score for an element to be pruned from the beam.

Path Beam Changes

The path beams are now composed of two structures; a "working beam" and a "post beam." The working beam is a sparse array (the size is based on the number of edge-lists in the grammar). This reduces parse collision searches to an order one search (each position in the working beam corresponds with a position in the grammar). The working beam is capable of maintaining all possible parsing paths. These large beam arrays are thus advantageous for doing exhaustive searches through the grammar. The working beams span a window of the maximum (allowed) length in segments of a word. After a working beam is processed and emptied, it is then rotated, or recycled, to the end of the active search window.

Each working beam can hold as many elements as there are positions in the grammar. When a working beam is completed (when no more words can end at that segment), its elements are down loaded into the post beam, a compact array much like the beam structures of the previous version. It is the post beam that is partitioned and maintains the top beamwidth elements.

The post beam structure is very flexible in terms of its maximum size per segment; it is no longer dependent on a uniform beamwidth. The new Decoder certainly takes advantage of this, as this feature is the highlight of the speed/accuracy improvements seen with this new version. With this version the Decoder will adjust the beamwidth with respect to the segment being processed, as follows:

```
Segments  0 -  6:  2.5 * beamwidth
Segments  7 - 14:  2.0 * beamwidth
Segments 15 - 19:  1.5 * beamwidth
Segments   >= 20:  1.0 * beamwidth
```

Also taking advantage of the flexible beam sizes, the post beam will retain any element with exactly the same score as its worst scoring member after pruning. Thus ties are no longer decided randomly for the path level.

New Slider Settings

The slider settings have been re-tuned to take advantage of the dynamic beamwidth capabilities. The speed/accuracy improvements are due almost exclusively to the sentence initial beam expansions. The path beam expansion will only be relevant to larger, more complex, grammars. For simple grammars there will probably be no speed or accuracy improvements with this change.

Conclusion

The primary differences to be seen in the new version are the speed and accuracy improvements from the dynamic beam sizes and parameter setting tasks. The new slider settings are tuned primarily for large grammars (many linear rules).

3.1.1.3. RELATIVE THRESHOLDING IN THE SEGMENT SCORING MODULE

INTRODUCTION

The objective of this project was to make this inner-most scoring process as independent as possible from the outer, syntax managing functions. By doing so the scoring functions can be logically separated from the syntactical transcription management functions; thereby greatly reducing the communication required between these two processes. This is desirable for any upcoming parallel implementation of the Phonetic Decoder Kernel.

BACKGROUND

Thresholds are used in the scoring process to eliminate poor class vs. phonetic code scores. Scores and threshold values are accumulated across acoustic-phonetic (PE-output) segments; if the threshold is set at 5, then at the second segment the threshold will be 10, at segment 4, 20. The threshold insures that the final per-segment score of an utterance will be at least that of the threshold. (This base threshold is also called the absolute threshold, when contrasted with the relative threshold.)

Final per-segment transcription scores tend to be far above the threshold. However, raising the thresholds to levels closer to average final per-segment transcription scores have proven disastrous to accuracy. A high threshold will never allow a transcription to cross a "glitch," an area of erroneous segmentation or classification.

Since scores are accumulated, thresholds become less meaningful when they are compared with good scoring words. If a transcription is scoring on average 40 per segment, and the threshold is 5, by the 10th segment the accumulated word-score will 400, and the threshold

will be 50. Considering that the worst score a class/phonetic code match can produce is -128, then at this point, any class, however poorly it may score, will be acceptable by the threshold. Transcriptions with high per-segment scores are then able to collect garbage, which can only be removed by beamwidth partitioning (a CPU intensive function).

Relative thresholding was implemented to watch the progress of a word and to adjust the threshold accordingly. The goal of the relative threshold is to maintain an adequate threshold for each word scored. By doing so the word scoring module need not use the constant threshold used by the transcription (syntax) management module, so it can be functionally independent of that module.

IMPLEMENTATION

The Relative Threshold Offset

The relative threshold is set according to the worst scoring element in the beam. Thus the threshold cannot be adjusted until the end of a segment scoring iteration. The relative threshold uses the worst score as an indicator of what will happen on the next iteration (which is the following segment).

In addition to the worst score, the relative threshold makes use of an "offset"; an adjustable variable that refines the relative threshold's prediction of the next segment's scores. The offset tunes the relative threshold's impact: negative offset values will allow a path scores to decrease, and positive offset values will only allow path scores that increase. The relative threshold is set accordingly:

Relative Threshold = Worst Score + Offset;

The offsets used in the implementation experiments were based on tests that measured the changes to the worst score (as the score progressed from segment to segment). The test showed the most common changes to the worst score. In addition it showed the percentage of accepted beam elements that would have been eliminated by a relative threshold that used the corresponding offset. The following table is an excerpt from these tests.

Offset	Observed	% of observed data accounted for at this offset
-8	74056	40.44%
-7	74509	41.49%
-6	76378	42.57%
-5	79939	43.70%
-4	79474	44.82%
-3	79597	45.94%
-2	81264	47.09%
-1	83903	48.28%
0	82606	49.44%
1	79474	50.57%
2	77075	51.66%
3	75396	52.72%
4	72543	53.74%
5	72022	54.76%
6	71246	55.77%
7	68855	56.74%
8	68021	57.70%

The table data is based on data from segment beams that required partitioning (the algorithm that finds the N best scoring elements to keep). Relative thresholding implements a pruning function, and ideally would eliminate the necessity for beamwidth partitioning entirely.

As stated the relative threshold sets itself according to the worst element in the (previous) beam. Yet since the relative threshold acts as the pruner, it would only be set when the beam filled, thus requiring partitioning. If the beam still has space then the relative threshold yields to the absolute threshold, thereby allowing the word in transcription a chance to fully utilize its allocated resources.

For experimental purposes we implemented a version that set the relative threshold even if the beam did not fill. In another implementation, when the beam did not fill, the relative threshold would yield to the absolute threshold only if the absolute threshold had a greater value; otherwise the relative threshold would keep its value.

Regulating the Grace Period

By default each word scored by the decoder gets a scoring "grace period" of three segments. During the grace segments thresholds are unused. Without thresholds the beam fills quickly with junk, which the partitioner removes later. In past tests the grace period has shown to be necessary for high accuracy. Relative thresholding is apt to reduce the needless processing of garbage scores that plague the grace period by monitoring and keeping with the worst score. During the grace period, if the beam does not fill, rather than yielding to the absolute threshold as it does normally, it would set itself to a value low enough that any score would pass.

Relative Thresholding Results

The following test results are all based on decoding 250 utterances with model 1078. Decoding performance results of the 6-Sept-88 version of the PDK are supplied for comparison.

Note on reading the tables:

"Word Acc"	word accuracy as a percentage of all words spoken.
"Utt Acc"	percentage of sentences decoded with no errors.
"+ Off 1"	percentage of sentences with one or less errors.
"+ Off 2"	percentage of sentences with two or less errors.
"Ave Time"	average decoding time in seconds/utterance.

Test 1: Relative threshold is set only when the beam fills. When the segment beam does not fill the relative threshold yields to the absolute threshold (as defined by the slider setting).

Decoder Version	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
-- slider 4 ----					
6-Sept-88	81.40	62.40	78.00	80.00	20.94
R.T w/ offset 0	83.87	64.80	81.20	83.20	20.50
-- slider 7 ----					
6-Sept-88	86.33	67.20	83.20	85.20	43.00
R.T w/ offset 0	86.42	68.00	84.40	85.60	40.05

When the relative threshold yields, the processing would return to its absolute thresholding. Measurements showed that this implementation only reduced the beam

element insertions and partitioning by 3%. This happened because the relative threshold was effectively being implemented only on every other segment. It would wait for the beam to fill, and then set itself. On the next iteration, because of the relative threshold, the beam would not fill and thus processing would return to absolute thresholding.

Test 2: Relative threshold is set only when the beam fills. When the beam does not fill the relative threshold retains its value. Only if the absolute threshold is higher will the relative threshold take that value.

Decoder Version	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
-- slider 4 ----					
6-Sept-88	81.40	62.40	78.00	80.00	20.94
R.T w/ offset 0	82.63	64.00	79.60	82.40	19.77
R.T w/ offset 5	82.77	64.40	78.80	81.60	19.55
-- slider 7 ----					
6-Sept-88	86.33	67.20	83.20	85.20	43.00
R.T w/offset 0	86.14	68.00	83.20	85.20	35.72
R.T w/offset 5	86.55	67.60	83.20	85.60	34.86

The speed improvement is minor at setting 4 because the absolute threshold at that setting is high enough that partitioning does not occur as often despite the relative threshold.

Test 3: Relative threshold sets itself on every iteration regardless of space in the beam.

Decoder Version	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
-- slider 4 ----					
6-Sept-88	81.40	62.40	78.00	80.00	20.94
R.T w/ offset 0	83.59	63.20	79.60	83.20	21.96
R.T w/ offset 5	82.50	64.00	78.80	81.60	20.19
-- slider 7 ----					
6-Sept-88	86.33	67.20	83.20	85.20	43.00
R.T w/offset 0	86.14	66.40	82.40	85.60	32.29
R.T w/offset 5	85.51	66.00	82.80	84.80	29.15

The slow down at setting four is caused by the overhead of having to scan the beam for the worst element when normally no extra processing is done. At setting four the partitioning requirement is low, which reduces the effect of the relative threshold, and so there is no benefit for the overhead of having to scan the beam. At setting 7, however, the partitioning requirement is high enough that even with the overhead of scanning the beam this extra thresholding is able to greatly reduce decoding time.

Test 4. This test has the same configuration as test 2, with the addition that relative thresholding was implemented during the grace period. In the grace period the relative threshold would set itself only when the beam was full.

Decoder Version	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
-- slider 4 ----					
6-Sept-88	81.40	62.40	78.00	80.00	20.94
R.T w/ offset 0	83.41	65.20	79.60	82.00	17.93
R.T w/ offset 5	82.36	63.60	77.60	80.40	17.74
-- slider 7 ----					
6-Sept-88	86.33	67.20	83.20	85.20	43.00
R.T w/offset 0	86.55	68.40	83.60	85.60	32.50
R.T w/offset 5	86.05	67.20	83.60	85.60	31.85

Using relative thresholding in the grace period removes enough of the garbage collected during this time to improve the speed without seriously affecting accuracy.

Conclusion

The relative thresholding, which allows the scoring module to use its own thresholding, making it a more complete and isolable module, is most beneficial in terms of speed/accuracy using the following implementation:

* Implement relative thresholds during the grace period.
* Set and update the relative threshold only when the beam fills.
* Only use the absolute threshold when it has a greater value than the relative threshold.

Though implemented specifically for parallel processing, the relative thresholding enhancement yields an improvement of 10-25% in speed and 2-3% in accuracy.

3.1.1.4. SIMPLIFICATION OF BETWEEN-WORD COARTICULATION PROCESSING

Introduction

Peter Ladefoged defines coarticulation (in his book *A Course in Phonetics* [Ladefoged 82]) as "the overlapping of adjacent articulations." He continues, "English consonants often vary their place of articulation so that they become more like the next sound." Previous releases of the decoder (with the dictionary) have tried to account for this linguistic phenomenon. This section covers the implementation and testing of several different schemas for doing between-word coarticulation in the decoder.

The dictionary itself accounts for coarticulation within a word. For between-word coarticulation the dictionary graph is set up to only allow certain transcriptions based on the condition that the previous word ended with a class belonging to a certain coarticulation class. In a coarticulated dictionary there can be several distinct initial transcriptions for any one word; the one chosen will be according to the (ending) coarticulation class of the previous word.

When a word completes a successful transcription, the ending class that brings the highest score for the word is the class saved for coarticulation context when scoring following words. The initial action in scoring a new word is to coarticulate it with its previous word.

Coarticulation involves two steps: the first determines which transcriptions can be used given the context of how the previous word ended. The second step involves the "free jump." If the word-final class of the previous word is the same as the word-initial class of the following word, then a free jump is attempted, allowing the new word to share the final segment of its previous word, in effect allowing it to begin with the set of second (rather than initial) classes.

In preparing the scoring module for network-parallel processing we have found that this current schema for coarticulation would be too cumbersome to support. It would require that the word scoring module be passed a set of coarticulation contexts in addition to the actual words to be scored. Similarly the scoring module would have to pass back, upon successful transcription of a word, the coarticulation context for the word just scored. In addition, this method for coarticulation would not allow for one final code preparation project, "Unique Word Scoring," which would allow the decoder to maximize the

efficiency of parallelism by never having the same word scored (starting at the same segment) more than once.

Background

The set of coarticulation information used to score a word consists of the word-final class of the previous word, and the coarticulation class of that word-final class. The coarticulation class is used with the dictionary to determine how the following word can be transcribed (with a non-coarticulated dictionary this step is not done, thus a word can use all transcriptions regardless of context). The word-final class is used in the "free jump", which enables a word to share the ending segment of the previous word if that previous class matches a word-initial class.

The Most Likely Class

In an effort to avoid the need to store, process, and communicate (for network parallelism), the decoded word final class (used for coarticulation), we decided to test the effect of using the most likely (ML) class of a word's ending segment (rather than the actual decoded class) as the basis for coarticulation. The most likely class of a segment is determined from the phonetic codebook, which gives the likelihood scores for all classes at all phonetic codes. The following tests show the results of decoding with the ML class in place of the word ending class of the previous word, as compared to the baseline which retains the actually decoded word ending class.

```
Data Set #1: syntax lung.nec, model 1078, 239 utterances.

test           |Word Acc| Utt Acc| + Off 1| + Off 2|Ave Time|
baseline       |  88.40 |  72.38 |  81.59 |  86.61 |   9.80 |
Use ML class   |  90.37 |  75.73 |  84.10 |  87.87 |  10.04 |

Data Set #2:  syntax atb4.nec, model 1007, speaker 1022, 250 utts.

test           |Word Acc| Utt Acc| + Off 1| + Off 2|Ave Time|
baseline       |  73.47 |  49.20 |  68.40 |  75.60 |  18.12 |
Use ML class   |  76.03 |  52.00 |  73.20 |  78.00 |  19.11 |
```

(lung.nec is an example medical dictation grammar, built to simulate a dictation task. atb4.nec is a standard test case grammar, built to be a difficult test.)

The use of the ML class for coarticulation not only seemed benign but it also improved the accuracy.

Free Jumps

Whether the ML class or the previous class is being processed the decoder carries around this baggage for use in the free jump. Currently to have a free jump, the previous (or ML) class can only match with word-initial classes that follow the specified coarticulation class (named by the previous (or ML) class's coarticulation class). We thought though, that this may be wrong. We wanted to see the effect of allowing free jumps to be independent of coarticulation (which is what happens with a non-coarticulated dictionary).

The following shows the results of decoding with free jumps independent of coarticulation.

Data Set #1: lung.nec, model 1078, 239 utterances.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
baseline	88.40	72.38	81.59	86.61	9.80
ind. prev	88.73	74.48	81.59	84.94	10.56
ind. ML	90.50	76.57	83.68	86.61	10.75
ind. both	90.04	75.31	83.26	87.87	10.29

Data Set #2: atb4.nec, model 1007, speaker 1022, 250 utts.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
baseline	73.47	49.20	68.40	75.60	18.12
ind. prev	74.38	50.40	71.20	76.80	19.50
ind. ML	73.52	50.80	70.40	75.20	19.65
ind. both	75.25	51.20	71.60	77.20	19.16

Key to above tests:

"ind. prev" => Free jumps independent of coarticulation: Use the previously decoded class for the jump.
"ind. ML" => ... Use the Most Likely class for the jump.
"ind. both" => ... Use both the Most Likely and the previous class for the free jump.

The independent free jump shows a difference, but not a significant one, especially if compared to the "Use ML class" test. Since the independent free jump gave dubious results we decided to test the the overall value of the free jump by not doing it at all.

The following shows the results of decoding without the free jump (this was named "never jump" -- hence the abbreviated name "never").

Data Set #1: lung.nec, model 1078, 239 utterances.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
baseline	88.40	72.38	81.59	86.61	9.80
ind. ML	90.50	76.57	83.68	86.61	10.75
never prev	88.34	72.38	81.17	86.61	9.56
never ML	89.65	74.90	83.26	87.45	10.02
never both	89.38	74.90	82.85	87.03	9.88

Data Set #2: atb4.nec, model 1007, speaker 1022, 250 utts.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
baseline	73.47	49.20	68.40	75.60	18.12
ind. ML	73.52	50.80	70.40	75.20	19.65
never prev	73.56	50.80	69.20	75.20	17.69
never ML	76.53	53.20	72.80	78.40	18.95
never both	75.16	52.00	70.80	76.80	18.30

In these tests, only the coarticulation class of the stored class (the Most Likely, the previously decoded, or both) is significant. With both data sets "Never ML" does very well compared to its counterpart "ind. ML," which means to us that the free jump really doesn't do anything important. In fact for the data set #2 (using generic model 1007) the "never ML" shows the best overall performance.

Conclusions

Since using the ML class in coarticulation does just as well as using the decoded class (if not better), and since using the ML class greatly simplifies the processing problems for network parallel processing, we decided to implement the use of the ML class and drop

altogether the handling of the previous decoded class. And since the free jump was in effect doing nothing, it was removed.

With a finalized method for coarticulation processing and relative thresholding, the decoder was optimized to take full advantage of the changes. These are the final results of "Never Jumping," using the coarticulation class of the Most Likely class with relative thresholding.

Data Set #1: lung.nec, model 1078, 239 utterances.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
baseline	88.40	72.38	81.59	86.61	9.80
the next PDK	89.65	74.90	83.26	87.45	8.79

Data Set #2: atb4.nec, model 1007, speaker 1022, 250 utts.

test	Word Acc	Utt Acc	+ Off 1	+ Off 2	Ave Time
May-88 PDK	72.47	50.00	67.60	73.60	31.41
Sept-88 PDK	74.20	48.80	69.20	76.00	23.14
baseline	73.47	49.20	68.40	75.60	18.12
the next PDK	76.25	52.80	72.40	78.00	16.42

The results indicate that the best implementation to use is the "Most Likely Class" (as defined by the phonetic codebook) as the basis for between-word coarticulation, and also to discontinue using free jumps. Based on tests with generic model 1007 this implementation, together with relative thresholding, yields a 29% speedup and a 2% accuracy improvement over previous releases of the Phonetic Decoder Kernel (PDK).

By using the "Most Likely Class," neither the transcription nor scoring modules need process the decoded class for coarticulation. This further reduces the information dependency the word scoring module has on the path transcription (syntax) module since it can determine the most likely class on its own. Also, since the free jump is no longer implemented, the last class of the previously decoded word need not be stored by the transcription manager, nor does it need to pass this information back to the scoring routines, thus simplifying the communication between the two modules.

3.1.1.5. THE UNIQUE WORD SCORING PROJECT

This project eliminated the scoring of words more than once starting at a given segment. As such it eliminates some excess work performed by the decoder. This improves the current non-parallel implementation of the decoder with a grammar dependent speed-up, as well as preparing for future parallel implementations.

I. UNIQUE WORD SCORING DESIGN AND SPECIFICATION

INTRODUCTION

Implementation of "Unique Word Scoring" reduces redundant (and needless) work done by the Phonetic Decoder Kernel (PDK). Unique word scoring makes it so that a given word will be scored only once at any one starting segment. The amount of word duplication will reflect the grammar: some grammars may have no duplications while others (such as the finite digit grammars) can cause duplication as high as 70%.

Unique word scoring, of course, will have a CPU overhead, though we do not expect it to be very significant. Nevertheless we can expect to see some speed degradation in grammars that have virtually no duplication, and a considerable speed increase for those grammars having high duplication.

Unique word scoring will invariably cause slight changes (plus or minus) in accuracy for some grammars. This project will affect which words are scored in parallel causing, because of unforeseen randomness, the accuracy change. However, compared to a run that only scored one word at a time (typically two words are scored in parallel) there should be no effect on accuracy at all.

UNIQUE WORD SCORING -- IMPLEMENTATION SPECIFICS

Memory usage was a major consideration for the design of unique word scoring -- the goal being to use as little as possible. The grammar (NEC) file itself already serves as an adequate data base for storing the connectivity between words. And the decoder's existing word beam structure stores all the other relevant information. With that in mind, the following design was made.

The word scoring process begins at the word beam. The current methodology processes each beam element independently. With unique word scoring, we add two steps to scoring a word:

1) Before scoring a word, the beam is scanned from the beginning to the current element. If any other (previous) element calls this same word then the scoring is not performed, no other action occurs; otherwise the word is scored as usual.

2) Upon completion of scoring a word, the beam is searched forward to the end to see if this word occurs for any other element. If the word is found then the scoring results of the word will be attached also to that element.

For an example consider the following word beam situation:

elem#	Beam	Following Words
0	|*| ->	a b c
1	|*| ->	d
2	|*| ->	b c
3	|*| ->	a

We start with beam element #0, and its first word 'a'. Before scoring 'a' we search all previous words for an occurrence of 'a'. And this being the first word for the beam, no occurrence will be found and the word will be scored. Next, upon completion of scoring 'a', we search forward through the beam, looking for an occurrence of 'a'. At element #3 we will find 'a', and so attach the results to this position (in addition to element #0).

We then score the next word of element #0, 'b'. Upon scoring 'b' we would find a forward occurrence of this word at element #2, and would then attach the scoring results to it.

When we get around to word 'b' of element #2, before scoring the word we would search backward to the beginning. We would find an occurrence of word 'b' at element #0, and so the word would not be scored, and we would continue with the next word.

OPTIMIZATIONS

As it stands this implementation requires a lot of traversing through the beam. The amount of searching can be reduced by doing one initial scan to check for the occurrences of duplication in the beam. With an external storage table, the size of which is parallel to the NEC word list, we can mark words that were found to be duplicate. With the same example:

elem#	Beam	Following Words
0	|*| ->	a b c
1	|*|->	d
2	|*|->	b c
3	|*|->	a

Word	Duplication Table
a	|2|
b	|2|
c	|2|
d	|1|
e	|0|

The initial scan loads the word duplication table: words found to be duplicate are marked in the table with the number of times they occur. When a word is to be scored the table is first checked. If the table entry for that word is 1, then we know the word has no duplicate and traversing the beam need not be done. With element #1, no traversing would take place with 'd', since the table would not have it marked as duplicate. Once a word is attached to a beam element, the value in the word duplication table is decremented. Thus when all the words are looked up and added to their appropriate beam elements, the duplication table will have been zeroed out in preparation for the next word beam.

For the rest of the processing of this beam, a value of zero will mean "already scored." Thus when 'a' is found following element #3, the duplication table will have a value of 0 for 'a' because 'a' has already been scored and added to all beam elements that it can follow.

UNIQUE WORD SCORING -- SPECIFICATION SUMMARY

Memory requirements: Dependent on grammar (one byte per word in grammar wordlist).
Speed improvement: Dependent on grammar (estimated 25% speed up on lung.nec, a medical dictation grammar).
Accuracy: No impact (expect some plus/minus changes due to randomness).

II. UNIQUE WORD SCORING IMPLEMENTATION TEST RESULTS

OVERVIEW

Unique word scoring has been successfully implemented into the phonetic decoder kernel. Test results show a significant speed increase, the extent of which varies with the syntax. Also, as projected in the initial design document, a slight change in accuracy was detected (an increase) due to the implicit changes made to which words are scored in parallel. However, decoding test comparisons run with single-word scoring (no parallelism) show that unique word scoring had no effect on accuracy (which was to be expected).

VERIFYING CORRECTNESS

From the design specification it was construed that unique word scoring would only affect decoding by mixing the words scored in parallel. By decoding with single-word scoring (one word scored in parallel) there should be no change in accuracy at any level (including per-word scores and incorrect utterances) when compared with a baseline that was decoded in the same manner. The following results demonstrate that this in fact is happening:

Single-Word Scoring Test #1)

Grammar: lung.nec (a medical dictation grammar)
Speaker Model: 1078
Utterances: 250
Baseline PDK: 3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	92.04	89.70	77.60	86.00	89.60	13.33
Unique	92.04	89.70	77.60	86.00	89.60	10.86

Single-Word Scoring Test #2)

Grammar: nums_infinite.nec (infinite digit syntax with "point")
Speaker Model: 1011
Utterances: 250
Baseline PDK: 3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	91.81	78.63	43.20	73.60	85.60	3.24
Unique	91.81	78.63	43.20	73.60	85.60	1.32

Single-Word Scoring Test #3)

Grammar:	nums_finite.nec (like nums_infinite.nec but constrains number of output words)
Speaker Model:	1011
Utterances:	250
Baseline PDK:	3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	92.11	80.56	43.20	75.60	89.20	6.67
Unique	92.11	80.56	43.20	75.60	89.20	1.52

Further comparisons showed that all internal scores were identical in all three single-word scoring tests.

MEASURING THE OVERHEAD

Unique word scoring's speed enhancements will depend on the internal structure of the grammar. Grammars with no duplication will not benefit from unique word scoring, and thus we would expect a slow down due to the overhead of the extra word-beam management. The following test is to measure the CPU time cost of unique word scoring. The decoder was run in a special forced-correct-mode for this test. This mode utilizes utterance-specific grammars which are generated from the (known) correct utterance text. These utterance-specific grammars accept only the correct text of that utterance. These grammars will yield no word duplications, and thus this test will measure the overhead due to unique word scoring.

Grammar:	Sentence-Specific
Speaker Model:	1078
Utterances:	250
Baseline PDK:	3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	98.94	98.33	97.60	98.00	98.00	2.99
Unique	98.94	98.33	97.60	98.00	98.00	3.00

The unique scoring code would appear to be a virtually CPU time free functionality, from this test anyway. Whatever the cost, however, this test does demonstrate that the unique word scoring code is an inexpensive functionality.

TEST RESULTS

Unique word scoring was tested using the default slider setting which scores two words in parallel. Testing was done on a Sun 3/75.

Test #1)

Grammar: lung.nec (a medical dictation grammar)
Speaker Model: 1078
Utterances: 250
Baseline PDK: 3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	90.42	88.71	76.00	84.40	88.00	9.34
Unique	91.73	90.05	76.00	86.00	89.20	7.56

Speedup = 19%

Test #2)

Grammar: nums_infinite.nec (infinite digit syntax with "point")
Speaker Model: 1011
Utterances: 250
Baseline PDK: 3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	91.62	78.60	43.20	73.20	86.00	2.11
Unique	91.62	78.60	43.20	73.20	86.00	0.96

Speedup = 55%

Test #3)

Grammar: nums_finite.nec (like nums_infinite but constrains length)
Speaker Model: 1011
Utterances: 250
Baseline PDK: 3-27-89

PDK VERSION	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
Baseline	91.81	80.58	42.80	76.40	89.60	4.60
Unique	91.91	80.67	43.20	76.00	90.00	1.09

Speedup = 76%

CONCLUSIONS

Yielding a significant (grammar dependent) speedup and showing (slight) positive changes in accuracy, unique word scoring is a successful project. It allows us to eliminate the duplication of word scoring effort at the start of a segment. This will also eliminate duplication of word scoring effort of words being scored in parallel, and is a necessary step in preparation for any parallel decoding implementation.

3.1.1.6. SINGLE-LEVEL BEAM SEARCH DECODING PROJECT

Overview

SSI originally used a two-level (path and segment) beam search as its underlying decoding method. Experiments with a one-level beam search indicated that a large decoding time improvement (a reduction to about 1/4 of the current time, a speedup of 75%) was possible by changing to a one-level approach, with no reduction in accuracy. The first part of this document shows a comparison of the one-level and two-level beam search methods, explains why we used a two-level method, gives a possible one-level implementation suitable for SSI's needs, and outlines some possible results of changing to a one-level approach. The second part of this document describes the results of some early tests with an initial implementation of a single-level version suitable for SSI's speech recognition system.

I. POSSIBILITIES FOR ONE-LEVEL BEAM SEARCH DECODING

Introduction

Most current speech recognition systems use some variation of a beam search method for decoding. A beam search method searches the several best (highest scoring, best matching) paths in parallel. This is a compromise between searching all possible paths in parallel and searching only the single best path. A beam search maintains a list of the previous best states (corresponding to partial paths), from which it derives a list of the current best states. Then the list of the current best states becomes the list of the previous best states, and so on, until the utterance is completely decoded.

Prior to deriving the current best states list from the previous best states list, the previous best states list is pruned, or cut back in some way to limit the expansion of the number of paths. Only the best paths are retained after pruning. Beam searches are usually segment synchronous so that the paths competing during pruning have comparable scores, since they correspond to partial paths over the same range of segments.

Beams can be constructed (at least) for any level of the language model. We currently use a two-level approach, with beams for both the segment and path levels. We have been using a one-level beam search version for some HMM experiments, where the one-level operates only at the segment level. This decoder is significantly faster than our two-level implementation of an HMM decoder, for the no grammar/universal grammar case. (The test version of the one-level decoder cannot yet use a finite state grammar, so we can only compare on the no grammar / universal grammar case.) Both decoders yield very similar results, so the speed improvement of the one-level decoder was not gained from a loss in accuracy.

Both decoders have been investigated to see where they spend time, in order to decide on a new algorithm combining the best features of both. We have a unique opportunity to study two implementations of beam search decoders, with very different internal approaches. Some of that analysis is presented in this document.

Historical Perspective and Parallelism

Long ago (prior to 12/85) SSI's decoder was a one-level beam search. At that time we converted to a two-level approach. The motivations for making the conversion at the time

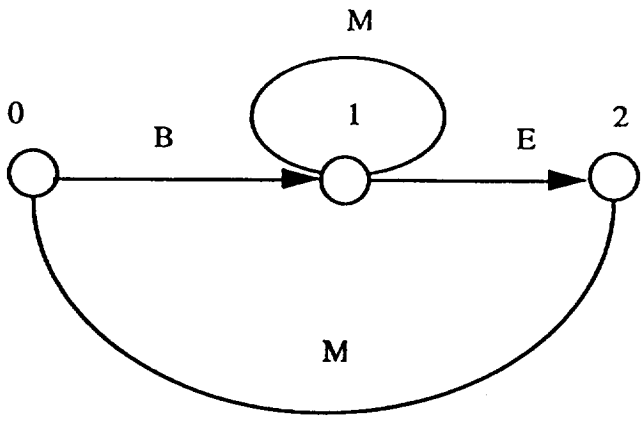
are now unclear, but the change resulted in a decoder which improved in both execution time and recognition accuracy.

We have had hints that an approach that scores many things simultaneously would be beneficial to decoder speed. The "Parallel Word Scoring" project grouped words into packages for scoring the packages in parallel. The words within one package are scored together, sharing the same beam space resources. When the words compete for the same resources, lower scoring paths that never amount to anything are removed earlier, thereby removing useless work from the decoder algorithm. The extreme conclusion of making alternate transcriptions compete for resources is the one-level approach. In a parallel implementation, the phones within words would be scored in parallel. Packages would be made of segment beam elements to score in parallel, rather than word packages to score in parallel.

The one data point from long ago has made us somewhat skeptical about changing back to a one-level approach. Most difficult to explain was why there was a speedup when we changed to a two-level approach, particularly when the new one-level approach was faster. Further investigation has revealed that using an absolute threshold to prune the beams was not implemented prior to conversion to the two-level approach. Pruning was accomplished only by limiting the number of partial paths in the beam. Thus changing to the two-level approach provided two beams, with two chances to prune due to beamwidth (the number of partial paths kept) limitations. This probably accounts for the reduction in decoding time when it was implemented. Now that we use many other beam pruning methods with a better understanding of their individual capabilities and interactions, a single-level approach may be faster.

A Time Comparison of the Two-Level HMM and the One-Level HMM Decoders

Some timing studies have been performed on the two-level HMM decoder and the one-level HMM decoder. The results below are for the following configuration: The dictionary is a single baseform dictionary. The HMMs are for a test model using HMMs with 4 transitions and 3 states. This HMM topology is shown in figure 3.1.1.6.1. The HMMs were generated and trained with a standard HMM trainer. The grammars used are given with the tests below. One utterance was used for these tests, which is: "who understands the fine print".



Phone HMM:

pdf B (Beginning) for transition (0,1)

pdf E (Ending) for transition (1,2)

pdf M (Middle) for transitions (0,2) and (1,1)

Figure 3.1.1.6.1. HMM Topology Used in Timing Tests

The search parameters chosen are the result of several quick tests, to configure the two-level beam search into a version more similar to the one-level beam search by scoring many words in parallel.

We compare the times for decoding this one utterance. We also compare the code profiles of the decoders' executions on this utterance, which gives the numbers of calls to various subroutines and the average time per call. All times are for a SUN-3/75, and should be more or less comparable.

Three configurations are compared in the following. The two-level HMM decoder with a "universal" grammar, the one-level HMM decoder with "no" grammar, and the two-level HMM decoder with a "by-example" grammar (a grammar built from a list of sentences, 650 in this case). These configurations will be indicated as follows:

two-level HMM Decoder with "universal" grammar:	2L-UNIV
one-level HMM Decoder with "no" grammar:	1L-NONE
two-level HMM Decoder with "by-example" grammar:	2L-BYEX

The 2L-UNIV and 1L-NONE tests are directly comparable. The 2L version using a grammar is given for comparison to see what the effects are on the statistics analyzed by using a grammar, which is our usual operating procedure.

Here we isolate several aspects of the decoding results for comparison.

Decoding Times:	(total sec.) optimized	profiling
2L-UNIV	953.56	1386.34
1L-NONE	237.25	321.30
2L-BYEX	40.60	60.50

This shows that the decoding times for 2L-UNIV are about 4 times as long as the decoding times for 1L-NONE. Also, the 1L-NONE times are about 5-1/2 times as long as 2L-BYEX. It's no wonder we use a grammar, since it also improves accuracy.

```
Number of Calls to and Time of Call of HMM Expansion Routine:
(Innermost Loop)
            Number Calls     Time/Call     Routine Name
                        (Average, self msec.)

2L-UNIV      6,599,055         0.08         classStateTransition
1L-NONE      1,638,532         0.09         expand_single_beam_elem
2L-BYEX        284,111         0.08         classStateTransition
```

The two routines compared here are almost identical in function; they perform the fundamental step of finding the following states of an HMM and putting the corresponding beam element into the next beam. The ratios between the numbers of calls is remarkably close to the ratios between the total decoding time: 2L-UNIV to 1L-NONE is about 4, and 1L-NONE to 2L-BYEX is about 5-1/2. Since the average time of a call is faster for the 2L version, the number of calls is clearly what makes the 1L version that much faster (and also the version using a grammar).

```
Number of Calls to and Time of Call of Dictionary Node Expansion Routine:
(Next to Innermost Loop)
            Number Calls     Time/Call     Routine Name
                        (Average, self msec.)

2L-UNIV      3,489,370         0.05         dictStateTransition
1L-NONE        671,477         0.04         dict_node_end_proc
2L-BYEX        161,599         0.05         dictStateTransition
```

The routines compared here are slightly less comparable, but still very similar in function; they find the following states in a dictionary graph when the end of an HMM is reached. The ratios of the number of calls are still similar, with the ratio 2L-UNIV to 1L-NONE being about 5 and the 1L-NONE to 2L-BYEX being about 4. Again the number of calls seems much more significant than the time per call.

```
Decoded Transcription and Per Segment Utterance Score:  (Total 92
segments)

            Per Seg Score      Transcription
2L-UNIV     -52111             wondered cancel fine print
1L-NONE     -52390             one extensive fine print
2L-BYEX     -55527             who understands the fine print
```

With all that extra searching going on, we wondered whether a better solution was being found. The 2L-UNIV version finds a higher scoring solution; whether its solution it really better is less clear, since both 2L-UNIV and 1L-NONE are wrong. The correct transcription, which was found by 2L-BYEX, has a lower score than the ones found by either 2L-UNIV or 1L-NONE, so maybe the problem in decoding this utterance is not in the decoder. Or, maybe this just shows the value in using grammatical limitations in the search.

Why Does a One-Level Beam Search Decode Faster? A Possible Explanation

Since the 1L decoder does as well as the 2L decoder while using much less time, accuracy has not been traded-off for speed. Thus a loss in accuracy is not the reason for its faster decoding. The simple and unenlightening answer comes from the statistics above: the one-level beam search expands fewer beam elements, both at the HMM and dictionary state levels. Thus it manages to do less searching. What we really want to know is then: what allows it to make fewer calls to these innermost functions?

(As a brief aside, the time involved in beam management overhead for the extra beams does not significantly alter the total decoding time, at least to the degree that the times for 2L-UNIV and 1L-NONE differ. The path beam management use and overhead account for at most about 5% of the total decoding time, as seen from the relevant subroutine times in the code profile output.)

Some reasons why the 1L version makes fewer calls to the inner loops can be deduced from the differences between the one- and two-level beam searches. In particular, the way in which partial path merging can take place is different, and may account for the extra effort used in the two-level approach. Partial path merging takes place whenever two different partial paths end up expanding into the same state. Most beam search algorithms will keep only one of the two paths in this case. This is because both will have the same possibilities for future matching, so the better one can be picked at the moment of their path merging, or collision. The worse one can be discarded since any paths following it can never be the best scoring overall path.

(This merging is a heuristic to find the best scoring overall path. It prevents us from finding the true overall top "n" best scoring paths, but gives us a better chance of finding the one best overall by making more room for paths that may get better later on. Finding the true best overall top "n" scoring paths is an algorithm variation that may be useful later on, in providing near misses for codebook training or alternate choices for output.)

Our two-level approach will only score words together (in a "parallel" package) that start from the same segment. This allows us to unique the words starting from a given segment. A word will only be scored once starting from a given segment. However, there can be no merging at the segment level between different instances of the same word that start at different segments.

The one-level approach does merge paths at the segment level between different instances of the same word (and grammar state) that start at different segments, but cannot guarantee that a word will only be scored once starting from a given segment. This seems to be a critical difference between the two approaches as far as limiting the search.

For the universal or no grammar case, there are no duplicate words starting at the same segment, since all words must start at all segments in the continuous speech environment. So word uniquing at starting segments provides no path merging. Furthermore, all words correspond to the same grammar state, so different instances of the same word starting at different nearby segments can all merge in path collisions. Merging paths for words starting at different segments will remove duplicate paths. Thus the one-level approach is the preferred method for the case with no grammar restrictions, since it does path merging which greatly reduces the search.

Most practical grammars do produce some number of duplicate candidate words to be scored starting at the same segment. When implemented, the unique word scoring code

that does this process resulted in a time reduction of about 20% on an example medical application grammar (the lung grammar), and a time reduction of about 75% on a grammar accepting a limited number of digits. This last case is one that is best suited for the unique word optimization, but will also have the same word starting at different segments, as long as there is any confusion about where a word ends.

The process of uniquing words starting at a given segment as a form of merging partial paths appears to be more grammar dependent than the other path merging method. Its usefulness depends on the internal structure of the grammar. The process of merging paths of the same word and grammar position starting at different segments is useful for all applications, as long as there is any indeterminacy about where a word ends. Finding word breaks is the essence of continuous speech decoding, and thus the merging allowable in a one-level beam search appears more effective in reducing the search. This different merging is what we think accounts for the time difference in the 2L-UNIV and 1L-NONE cases.

Word ending indeterminacy can be illustrated by viewing dumps of the path beams. In partial path beam dumps, we can see the same word ending at several consecutive segments. With the merging of different instances of the same word starting at different segments, the same words following these different endings could all merge together. This would reduce the amount of redundant searching taking place.

So the conclusion is that it is more important to merge points in the grammar than to unique words starting at a given segment. This is because it is a special case where multiple instances of the same word start at different grammatical states at the same segment, but it is a common occurrence when multiple instances of the same word and grammar position start at different close segments.

Outline of Possible Single-Level Implementation

Some notes are given here about how we envision a single-level beam search implementation given our other considerations (grammar, etc.). Some familiarity with the current system is assumed. This is given for the HMM version. The non-HMM version would be a simplification of this.

Beam Data Structures:

Segment and path level beams would be kept, similar to what they are now. The path level beam is kept primarily as a handy storage for word endings and for recovering the decoded word sequence, but will provide some pruning as it does now. The post-word-beam would be as it is now. The other main path beam structure would only need to be available for one segment, instead of requiring beams for the maximum number of segments in a word as is necessary now.

Only two segment beams are required, a current and next segment beam. The segment beam has a finite allocation. The segment beam cannot be allocated to the maximum required, since that would be an extremely large number (determined by the number of grammar edges, the number of words, the number of nodes in a word's sub-dictionary, and the number of states in an HMM). A collision at the segment level is determined by the grammar edge list, the dictionary word id, the sub-dictionary node, and the HMM state. These four items are used to form a hash index into the segment beam.

Initialization:

A post beam element is made corresponding to the initial node of the grammar. This goes into the segment-index-(-1) post beam.

Beam Search:

The current seg. beam is expanded into the next seg. beam. The post beam for the current segment is used to start words expanding into the next seg. beam. If the next seg. beam gets too big, it is pruned back to some tolerable size. When expansion is complete, the next seg. beam is pruned. Words ending in the next seg. beam are put into the (main) path beam. This is pruned and the post beam (for the next seg.) is built. The next segment becomes the current segment and the process is repeated until we run out of segments.

Pruning Methods:

All methods (known) will be available for beam pruning. Their application will be ordered with efficiency in mind. Eventually the search pruning parameters will be automatically trainable. Some of the pruning methods considered will be: finite size (beamwidth), absolute minimum threshold, relative threshold from maximum, relative threshold from previous minimum. Segment beam elements of the next seg. beam which can not expand (correspond only to word endings) are removed when the word ending beam elements are found. Beamwidth prunings are to be avoided so that partitions can be avoided as an ordering consideration in the application of pruning methods.

Other Possible Benefits

The one-level beam search approach is inherently time synchronous, and is therefore suitable for pipelining. A pipelined version of this algorithm makes much more sense than one of the two-level beam search.

Fewer beam levels implies fewer beams to prune, which further implies fewer pruning parameters to be selected. By keeping some vestigial path beams the path beam parameters will still be present but should not have as much of an effect on the decoding.

Several things we want to try become easier to implement with a one-level approach: code entropy based pruning and anything like that that happens once per segment.

Conclusions

The one-level beam search is more efficient than the two-level beam search because it allows for a type of path collision/merging that the two-level does not. This merging is more frequent than the mergings that the two-level can use that the one-level cannot. The one-level beam search is not appreciably different in terms of accuracy than the two-level beam search, so any one-level implementation by SSI is not expected to be appreciably different in accuracy. However, a one-level beam search can possibly yield a decoder 4 times as fast. An outline of a possible one-level implementation for SSI has been given.

II. INITIAL RESULTS USING SINGLE-LEVEL BEAM SEARCH PHONETIC DECODER KERNEL

Overview

An initial single level version of the phonetic decoder kernel (PDK) has been implemented, as described in the preceding document. Some results of tests comparing our current PDK and the new version are given. These show a remarkable improvement in decoding speed, and sometimes an accuracy improvement as well.

The single level version allows us to prepare for implementing a parallel processing version of the decoder that processes the innermost part of the algorithm in parallel, the phone matching portion. This is an obvious extension (in retrospect) of our former plans to implement word scoring in parallel.

Results

Some comparison tests between the new single level beam search decoder and the most recent two-level version have been run on a variety of grammars. These were all run on a SUN SparcStation1, so the times should be comparable. These tests use the data from the 3.2 release generic male model testing for dependent speakers, with independent data. There are tests using the lung grammar, the pmtext grammar, and two digits grammars.

Baseline test results are given for two internal search parameter settings, s4 and s8. (The settings are from a range of settings, s0 to s9. These settings capture a predefined accuracy vs. decoding time tradeoff: s0 being the fastest and least accurate, s9 being the slowest and most accurate, and s4 being the default optimum tradeoff point.) Single level test results are given for a number of values of a new parameter, the segment beam relative threshold from the maximum in the beam. In the single level version, this (new) relative threshold prevents expansion of the segment beam elements that don't score within this range of the maximum in the beam. Of course, not much experimentation of parameter settings has been done so far for the single level version.

The baseline results are indicated with the base-, and the single-level results are indicated with the sl- prefix in the test names below. The times are average SUN SparcStation1 decoding times per utterance (in seconds)

For the lung grammar tests:

test name	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
baselung-s4	91.59	91.59	79.35	87.65	89.98	1.47
baselung-s8	94.48	93.98	84.21	90.69	93.42	2.47
sllung-256	89.81	88.41	78.85	85.02	88.56	0.50
sllung-320	92.77	91.65	82.49	88.66	91.90	0.65
sllung-384	94.07	93.40	83.81	90.59	93.22	0.78
sllung-512	94.67	94.04	84.72	91.60	94.03	0.96

This shows, for example, that we can exceed the accuracy of the old setting s8 while decoding faster than the old setting s4.

The pmtext (bpm) tests:

test name	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
basebpm-s4	87.38	90.04	37.85	74.93	89.17	4.72
basebpm-s8	88.73	91.48	38.54	76.78	90.73	8.23
slbpm-256	78.80	80.29	34.93	67.32	79.32	1.07
slbpm-320	82.16	84.00	37.37	71.32	83.90	1.38
slbpm-384	82.90	84.70	37.85	72.00	84.49	1.66
slbpm-512	82.96	84.79	38.15	72.10	84.39	2.06
slbpm-1024	87.57	89.77	39.61	76.20	88.88	4.18

The single level version here is slightly less accurate, but is much faster, and is more accurate in some instances in utterance accuracy.

The digits test, using the factory standard semi-infinite grammar (only one occurrence of the word "point" is allowed):

test name	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
basedig-s4	93.73	86.61	56.54	85.11	94.77	0.13
basedig-s8	93.88	87.20	57.34	86.12	95.77	0.21
sldig-256	93.43	88.34	59.76	86.92	96.58	0.09
sldig-384	93.53	88.44	59.96	87.12	96.58	0.14
sldig-512	93.53	88.44	59.96	87.12	96.58	0.18

The results here would indicate that perhaps a more constrained search would give a better tradeoff for the single level version, which would still be more accurate than the old setting s8.

Another digits test was run to see what happens with a grammar where "unique word scoring" would help in the old two level decoder. Nothing corresponding to "unique word scoring" is implemented in the single level version. This is where words starting at a given segment are uniqued, i.e., are not scored more than once. (This would be much more complicated to implement in the single level version.)

The digits test, using a finite digits grammar (up to seven digits), where "oh" and "point" are considered digits (because they're in the data -- this uses the same data as the previous digits tests):

test name	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
basedig7-s4	93.58	88.32	60.76	86.92	96.18	0.15
basedig7-s8	93.68	88.33	60.16	87.32	96.18	0.23
sldig7-256	93.38	88.67	60.97	87.32	96.58	0.17
sldig7-384	93.43	88.72	60.97	87.53	96.58	0.26
sldig7-512	93.43	88.72	60.97	87.53	96.58	0.34

The results here are similar to the earlier digits test, but the "finite" grammar highlights the duplication of effort eliminated by "unique word scoring" in the two level version.

However, the time is probably not much of an issue in this case, although it looks like it can still be faster as well as more accurate than the old version.

Conclusions

The single level version definitely appears to be the way to go for a (varying by grammar) speed improvement, and often an accuracy improvement. Because the simple guesses at parameter settings given above produced generally better results, we can assume that settings better (in speed and probably accuracy) than the settings in the current version can be found. Recalibration of the decoding parameters is necessary before any delivery of the new system.

The single-level version also offers a better potential for a most efficient parallel implementation. With a single-level version, we can concentrate on making the innermost portion of decoding run in parallel. Furthermore, we have the benefit of earlier merging of duplicate paths, which eliminates duplicate effort in a sequential or a parallel version.

3.1.1.7. DECODER RE-ENGINEERING FOR PARALLELISM: SUMMARY

Introduction

Re-engineering the decoder in preparation for a parallel implementation has produced a very different decoder. The changes that were necessary have also forced us to rethink many of the original design considerations. This rethinking has yielded a much better decoder, not only because it is better prepared for a parallel implementation, but also in absolute terms of speed and accuracy. This section of the report compares the performance of all the decoder versions prepared for this project on three separate test applications.

Decoder Versions

There are five versions of the decoder which are the result of the various stages of the decoder re-engineering for parallelism task. These correspond to the changes whose reports were given earlier in this report: parallel word scoring, dynamic beam sizes, relative thresholding, coarticulation simplifications, unique word scoring, and single-level beam search decoding. (Relative thresholding and coarticulation simplifications were combined and released into the same version, so there are only five.) These versions will be indicated as follows:

par-word	parallel word scoring
dynam-bw	dynamic beam sizes
thr-coar	relative thresholding and coarticulation simplification
unique-w	unique word scoring
sing-lev	single-level beam search decoding

Comparison Tests

Three test applications have been used to compare the performance of the various versions of the decoder. These are: an example medical dictation grammar, the "lung" grammar; a digits grammar, the "dig" grammar; and a difficult generic test grammar, the "pmtext" grammar. There are 988, 497, and 1025 test utterances for these applications, respectively.

The utterances are essentially equally distributed among the eight speakers for each application.

The test applications correspond to no particular delivered version of SSI's speech recognition system applications. They have been developed to be more difficult than our usual delivered applications, primarily for research purposes. More difficult applications make changes in performance easier to see in the performance results, because with more difficult applications there is no saturation effect on the accuracy scores at the high (or the low) end.

The grammar (with a corresponding dictionary) was the only change in these tests other than the decoder version. The other components of the tests were the same throughout, in particular the speaker model, which was the 3.2 release generic male model.

Results

The following tables give the accuracy and decoding time results for the various versions of the decoder corresponding to the various stages of this project, as listed above.

Note on reading the tables:

"Word Acc"	word accuracy as a percentage of all words spoken.
"W Trans"	word accuracy as a percentage of all words transcribed.
"Utt Acc"	percentage of sentences decoded with no errors.
"+ Off 1"	percentage of sentences with one or less errors.
"+ Off 2"	percentage of sentences with two or less errors.
"Ave Time"	average SUN SPARCstation1 decoding time per utterance (in seconds)

For the lung grammar tests:

version	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
par-word	91.39	90.35	79.45	87.04	89.57	2.84
dynam-bw	91.20	91.18	79.45	87.75	90.38	2.21
thr-coar	90.36	90.31	78.24	86.23	89.47	1.68
unique-w	91.59	91.59	79.35	87.65	89.98	1.47
sing-lev	94.07	93.40	83.81	90.59	93.22	0.78

The digits tests:

version	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
par-word	94.04	86.25	54.53	84.91	95.57	0.58
dynam-bw	94.04	86.21	54.33	85.11	95.37	0.45
thr-coar	94.09	86.46	54.93	85.31	95.17	0.35
unique-w	93.73	86.61	56.54	85.11	94.77	0.13
sing-lev	93.43	88.34	59.76	86.92	96.58	0.09

The pmtext tests:

version	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
par-word	82.93	85.64	36.88	72.00	84.98	8.48
dynam-bw	83.20	85.55	36.29	71.51	84.29	6.17
thr-coar	83.26	85.58	36.10	71.12	84.49	4.57
unique-w	87.38	90.04	37.85	74.93	89.17	4.72
sing-lev	82.90	84.70	37.85	72.00	84.49	1.66

Discussion

For all test applications, we can see a steady improvement in both decoding time and accuracy. The single-level results are the most unusual in this progression, but that is because they are the most recent. The single-level version has not had its internal parameters optimized for best performance. This is also true of the unique word version, but that version is much closer to the other versions than the single-level version, so that not having its internal parameters optimized is less of an issue.

Conclusion

Decoder re-engineering for parallelism has produced a much different decoder, one that is significantly more accurate and much faster. The improvements are application grammar dependent, but fall in the range of about 1-5% absolute utterance accuracy improvement and about 75-85% decoding time reduction. Furthermore, the decoder is now structurally much better prepared for any upcoming port to a parallel processing environment.

3.1.2. PARALLEL PROCESSING PREDICTIONS

Naturally, we would expect a parallel implementation of our speech recognition decoder algorithm to result in a reduction in decoding time. But because of the improvements we have achieved in preparing the algorithm for a parallel implementation, is the possible speedup from a parallel port still worth the effort, and still worth using a rarer type of hardware? What are the specific issues and tradeoffs involved? These are the questions we seek to answer in this section.

We had originally intended to analyze possible parallel implementations in our "home" computer environment. When the contract was begun, this environment was a VAX-8800, which supports multi-processing and communication across dual CPU's in an asymmetric tightly coupled configuration. Our "home" environment has since changed to be an ethernet network of SUN Microsystems workstations, composed of SUN-3's, SUN-4's, and SUN SPARCstation1's. This environment supports multi-processing and communication across multiple CPU's as well. Many of the individual SUN CPU's are faster than the VAX-8800's CPU's, but the ethernet communication between SUN CPU's is slower than between the tightly coupled VAX-8800 CPU's. Our resulting predictions about a parallel implementation's performance depend on both the basic CPU speed and the communication overhead. The predictions will yield different conclusions for different computing environments.

In this second part of the final report, we first consider a parallel processing implementation plan. This shows how we might implement the re-engineered decoder to use a network of processors, similar to our own network of SUNs. Next we present a theoretical study of the potential decoding time reduction benefits of a parallel implementation. This estimates the speedup as a function of computation overlap and communication overhead time. Then there is a section on the measurement of the expected network communication overhead if we are to use our own SUN network. Finally, we make some conclusions about when a parallel implementation becomes worthwhile.

3.1.2.1. PARALLEL PROCESSING IMPLEMENTATION PLAN

Introduction

A plan for a parallel processing implementation of the final reengineered phonetic decoder is given here. Some details are included on how to get procedures to operate in parallel over a SUN network, such as the one available at SSI. The algorithm is given in a pseudo-code format, from which is should be clear how it may be implemented on similar target architectures.

Using a SUN Network for Decoder Parallelism

Remote Procedure Call (RPC) is a network programming system developed by Sun Microsystems. The following RPC discussion is based on information from the Sun Educational Services course entitled "Network Programming Using RPC."

With RPC, the phonetic decoder kernel can be spread across the network. The implementation plan is to set up some portion of the low level scoring function as an RPC "server" daemon (similar to the Network File System (NFS) and "Yellow Pages") on several hosts (the scoring function accounts for more than 90% of the CPU time spent by the decoder). The rest of the decoder will run on one host as the "client" and organize the remote calls to the scoring functions.

The normal behavior for RPC is to block the client calling process until the remote procedure responds with an answer. This situation would be unacceptable, since parallelism is the goal. Fortunately RPC allows the calling process to specify how long to wait for the result, and in this case the client will implement a "zero wait."

Since the client does not wait for a response from the remote scoring function, the communication link is lost, and thus the scoring function cannot return its results directly. The results will have to be returned by a second function from which the client process will wait for a response. This is one way we can achieve parallel processing in the decoder on a SUN network.

A Parallel Processing Phonetic Decoder Kernel Implementation

The original high level pseudo-code description of the single-level version of the decoder is as follows:

```
FOR EACH speech segment "X"
  FOR EACH partial path "P" active at segment "X"
      BEGIN
        score next possible phones of partial path "P";
        record results;
      END;
```

At the beginning of the partial path loop (the inner loop in the pseudo-code), we know how many active partial paths have been kept. This is the current active beamwidth. It is the expansion of these partial path states that we want to do in parallel. The parallelism achievable is limited by the number of processors available. We divide the active partial path states into equal size groups to be processed in parallel. We make as many groups as there are available processors. Then the processors are called, each with their own package of partial path states to be expanded, all to execute in parallel. The results are then returned to the client process, which puts the results together.

In pseudo code the client process will run as follows:

```
FOR EACH segment "X"
  BEGIN
  Build ("current beamwidth" / "number of available processors")
      groups of partial path states active at segment "X";

  /* FOR EACH group "Gi" to be sent to Host "Hi" */
  COBEGIN (IN PARALLEL)
      expand and score group "G1" on Host "H1";
      expand and score group "G2" on Host "H2";
      expand and score group "G3" on Host "H3";
      ...
      expand and score group "Gn" on Host "Hn";
  COEND (IN PARALLEL);

  record / merge together results;
  END;
```

Different parallel hardware environments will implement the parallel (COBEGIN-COEND) portion of the above algorithm differently. Taking into account the RPC method for making processes run in parallel, the parallel portion of the above pseudo-code would be expanded as follows for a SUN network such as the one at SSI:

```
FOR EACH group "Gi" (to be sent to Host "Hi")
    BEGIN
    RPC call: expand and score group "Gi" on Host "Hi", no wait;
    END;

/* ALL SERVERS ARE BUSY */
/* NOW GET RESULTS */

FOR EACH group "Gi" (running on Host "Hi")
    BEGIN
    RPC call: get-results for "Gi" from Host "Hi",wait for response;
    record / merge together results;
    END;
```

The server daemons will support the partial path state expansion and scoring function, which will store its results in a data structure global to the daemon. The expanding and scoring function will not return any results. A second server function, "get-results", will report the results of the expanding and scoring process.

The "record/merge results" portion of the code has been put into the second loop of the expanded COBEGIN-COEND part of the algorithm. This is to avoid keeping the returned results of all the sub-processes active in memory at the same time. The results of all the sub-processes must be integrated into one beam structure anyway. This way the results are put into the beam structure when they become available, possibly enabling parallelism with that process and the other sub-processes already executing.

Conclusion

The decoder can be ported to a parallel processing environment similar to a network of SUNs, by using a method such as the one given here. Few changes would be required beyond the reengineering tasks that have already been performed.

3.1.2.2. ESTIMATION OF ALGORITHMIC SPEEDUP FROM PARALLEL PROCESSING

Introduction

The change in decoding time due to implementing the decoder in parallel will be estimated in this section. The plan given in the earlier section for implementing the phonetic decoder kernel on a network of processors will be used as the implementation model. Time reduction estimates will come from estimates of the time spent in the different portions of the code; the unparallelizable part, the part duplicated in parallel, and the fully parallelizable part.

Unparallelizable Code

A small part of the code, in terms of execution time, will not be affected in our parallel implementation. This part is not parallelizable, and will be left as it is in our parallel implementation. This part of the code is mostly setup and post processing, preparing the input to be used by the decoder and preparing the output to be given to the user. It is worth explicitly stating that this part of the code does not run in parallel with any of the other portions of the code, that is, the unparallelizable part does not execute concurrently with the parallelizable part, etc. Perhaps the unparallelizable could be executed in parallel with the parallelizable part of different utterances, for processing multiple speakers or multiple utterances for the same speaker (with "talk-ahead"). That is beyond the scope of this analysis.

Duplication of Effort for Parallelism

As planned, the part of the decoder algorithm that will be executed in parallel is the part that does the extension and phone scoring of partial paths. This part is repetitive, having to be called once for each of the different partial paths to be extended. The different calls are all independent, making them suitable for execution in parallel. The beam to be extended is divided up among the available processors. Each processor then does the extension and phone scoring for a roughly equal fraction of the original beam.

Each sub-process also generates a following beam, consisting of the partial path states extended from its original list of path states (its fraction of the initial beam). A key part of building this structure is the elimination of duplicate entries, partial path states that are the same but have been generated as successors of different prior states. This may happen whenever there is a merging together of paths in the grammar or the dictionary. This part of the code forms a significant part in terms of execution time.

Duplicate state elimination takes place whenever a new path state extension is generated. Some of this merging will thus take place within the parallel sub-processes. But it will need to be repeated for all surviving path states returned to the main calling process, to be sure of eliminating duplicate states located in different returned sub-beams.

This part of the algorithm is thus a special case with regard to parallelization. It will be performed on the portion of the new beam generated by each sub-process within that sub-process (in parallel), and repeated by the main calling process on all elements of all returned new sub-beams. The repeated part cannot be performed in parallel with any other part of the code; it requires the output of all of the sub-processes.

Therefore, timing estimates must consider the beam element duplication elimination part of the code as being executed by both the main calling process and the sub-processes. Each individual sub-process will perform only a fraction of this effort, because they operate on only a fraction of the original beam. But the duplication of effort by the main process and the sub-processes will remain as a price for dividing the original effort among the sub-processes.

The repeated part will probably be less than a full duplication of the processing that would take place in a sequential implementation, as some duplicate state elimination may be completed in the sub-processes. By considering the time of this process as being fully duplicated, the estimates of the possible speedup available from the proposed implementation should be relatively conservative.

Fully Parallelizable Code

The remaining part of the code is the fully parallelizable portion of the code. This is the majority of the code in terms of execution time. It is fully parallelizable in the sense that pieces of it can be run in parallel with similar portions of the code, and no duplication of effort is required to put the results together. This part of the code may also be executed concurrently with the sub-pieces of the code duplicated in parallel, but not the non-parallelizable part that must be duplicated after the sub-processes are complete.

Network Communication Overhead

Some time is required to call the sub-processes on the different processors across the network, and to return their results. This effort is functionally similar to the time required for subroutine calls, except that the subroutines are now sub-processes executing in parallel on different processors. This extra time required by the parallel implementation of the decoder will be modeled as if it is similar to subroutine call overhead. There will be some (network) communication overhead time required that depends on the number of calls to sub-processes, the amount of data that must be passed to the sub-processes, and the amount of data returned from the sub-processes.

Parallelism Limitations

Since a processor cannot be given less than one partial path state, a limitation on the number of processors executing in parallel is the number of path states in the original beam to be expanded. Processors beyond this number will remain unused.

Speedup Estimates from Parallelism

The model of the parallel implementation of the decoder breaks up into four major parts with regard to execution time estimation: the part that is not parallelizable, the part that is fully parallelizable, the part that is duplicated in the parallel implementation, and the network communication overhead. From these four parts we can derive an expression for the execution time of the sequential version and the parallel version.

For this formulation we will estimate the time to process one segment of input (one phonetic code output from the PE). This is to provide a better model of the network communication overhead, which will be added to the total per segment time as many times as there are available sub-processors, one for each call. This is because the work for processing one segment is divided among the available processors at the start of processing that segment. For the total predicted decoding time, the time calculated here must be multiplied by the average number of segments in an utterance. By giving the estimate in this way, we can estimate the speed change independent of utterance length and application.

First, denote with the following variables:

U -- the unparallelizable portion's execution time per segment
P -- the fully parallelizable portion's execution time per segment
D -- the duplicated in parallel portion's execution time per segment

C -- the network communication overhead time per call
N -- the number of available sub-processors, including main routine's machine

T -- the total time of execution per segment

(The main calling program's machine is included in N since one of the sub-processes should be run there, given the division of code with respect to parallelization of our implementation.) Then an expression for the sequential decoder's execution time per segment would be:

$$T_s = U + D + P$$

Neither the network communication overhead nor the number of available sub-processors plays a part in this formula, since it is executing on only one processor. This formula is used with estimates from code profiling statistics to estimate the time spent in the different portions of the algorithm. These are then used for predicting the time of the parallel implementation.

An expression for the parallel decoder's execution time per segment would be:

$$T_p = U + D + D/N + P/N + C*N$$

(Note: The number of available sub-processors (N) actually used will not exceed the number of beam elements to be expanded. Thus if N is greater than the number of beam elements to be expanded, the value used for N for estimating the execution time will be limited to the number of beam elements to be expanded. This is usually not the case, as N may typically be less than 20, while the number of beam elements is typically greater than 100.)

A visual comparison of the per segment time required by the sequential and parallel versions of the decoder is given in Figure 3.1.2.2.1. This illustrates the overlapping (in time) portions of the code and the overhead required by the parallel implementation.

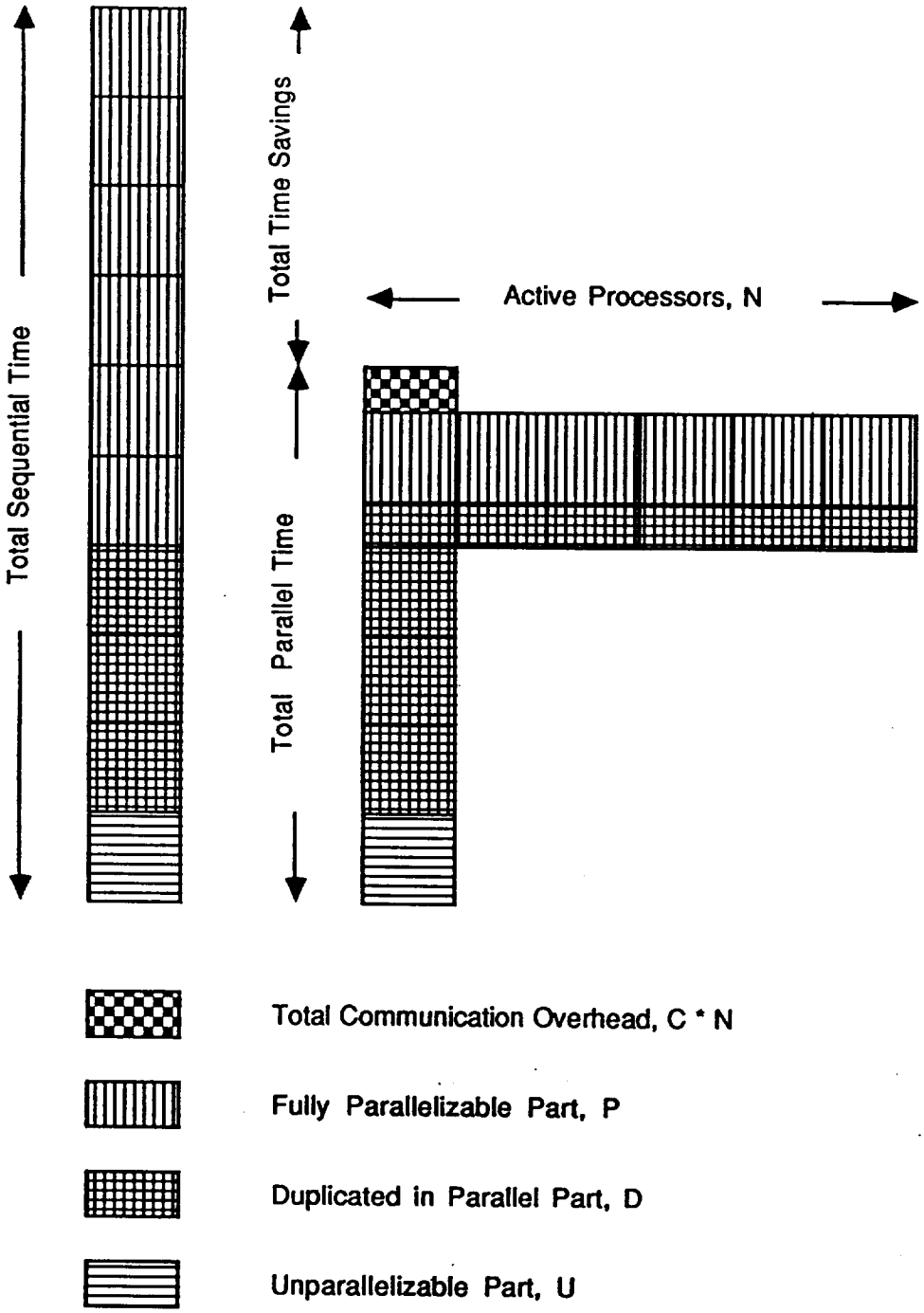


Figure 3.1.2.2.1. Visual Comparison of Sequential and Parallel Implementations' Decoding Time per Segment

Observations

This formulation shows several things about the planned parallel implementation. With the addition of one processor, the time may actually increase because of the duplicated part of the algorithm. Beyond this, the time decreases as N increases (if the network communication overhead is sufficiently small) until either the beamwidth limitation ends the progression, or the communication overhead term dominates and the time increases.

By disregarding the beamwidth limit on N (and assuming the communication overhead is essentially infinitesimal), a bound on the minimum execution time is produced by assuming the terms divided by N become vanishingly small as N becomes very large. This is useful in predicting the limits of the parallel implementation.

$$T_{min} > U + D$$

By examining the parallel time estimate formula, the duplicated part D stands out. This portion of the algorithm affects both the main calling process and the sub-processes. This is thus an important part to focus on when optimizing the original code. It is clearly the bottleneck of communication in the parallel implementation.

Conclusions

Further details have been examined for the proposed parallel implementation of the decoder, for estimating its execution time. Expressions for the execution time of the sequential and parallel versions of the decoder have been given. These expressions can be used to predict the parallel version's execution time from measurement of the sequential version's execution time, and from measurement of the network communication overhead.

3.1.2.3. ESTIMATION OF NETWORK COMMUNICATION OVERHEAD

INTRODUCTION

The network of SUN computers in use at SSI offers a resource yet to be utilized: distributed programming. A "distributed program" is a single application that performs various tasks, and sub-tasks, on separate machines that are linked together via a network (in SSI's case that network is Ethernet).

There exist many methods for distributed programming. The central criterion is communication between the remote nodes: the distributed processes need to communicate with each other. With the UNIX operating system the most immediate and convenient method for inter-process communication is with i/o sockets. The traditional socket, however, is intended for use on the same machine: the most common socket being the "pipe." SUN's UNIX supports a socket (called an AF/INET socket) that registers itself with the network and is accessible to any other linked machine. This is the type of socket we are interested in, as it allows for remote machines running independent applications to communicate with each other.

SUN offers an additional package that makes using these sockets easy to implement: the "Remote Procedure Call" (RPC) library. The high-level routines supplied in this library can be modified to the specific needs of an application. Due mainly to its (relative) ease of

use combined with its versatility, the RPC library was used as the distributed programming communication protocol for estimating the network communication overhead of a parallel distributed implementation of the phonetic decoder.

MEASURING RPC COMMUNICATION TIME

An application running as a distributed program will have obvious speed improvements over its linear counterpart, as it can execute functions in parallel rather than sequentially. Nevertheless there will be a communication cost, which may negate any possible speed improvement. What we want to know is how fast (or slow) is this remote process communication.

To measure this communication time, two programs were developed: the first, an RPC server that simply waits for input on its socket; and the other, an RPC client that sends messages to the server and then waits for a response. Neither the server nor the client performs a given task with the data being communicated; both merely send and receive data.

In the proposed parallel decoder implementation, the main program would be the client process. The client process would request processing resources from the available server processes, which would perform some local scoring subprocess of decoding.

This communication-only activity was timed from the client's perspective. Since the AF/INET socket communication is a blocking request, REAL time was measured (as opposed to CPU time). By measuring real time we find the actual time cost of the Ethernet for a remote call.

CONFIGURATION

The following communication cycle was measured:

Step 1. Client sends an ARRAY to the Server. The Server receives data and verifies; then sends one unsigned integer back to Client. Client verifies.

Step 2. Client sends NOTHING (just a signal) to the Server. The Server responds by sending an ARRAY; Client receives and verifies data.

The cycle was measured with the ARRAY configured in the following manners:

1. An Array of 25 short integers.
2. An Array of 25 long integers.
3. An Array of 100 short integers.
4. An Array of 100 long integers.
5. An Array of 250 short integers.
6. An Array of 250 long integers.
7. An Array of 500 short integers.
8. An Array of 500 long integers.

The cycle was also measured varying the means of sending and receiving the data:

1. Iteration: The array was sent/received one element at a time.
2. Vector Transport: The array was sent/received using the RPC routine 'xdr_vector' which processes entire arrays.

RESULTS

The server and the client ran on separate SUN SPARCstation1 workstations. The cycle in each of the different configurations was executed 50,000 times. The total time for all the cycles was measured, instead of measuring the time of the individual cycles and taking the sum. This total was then divided by the number of cycles to get the best time estimate while avoiding any potential problems with the timer's resolution.

TEST #1

In this first run, the time was measured in the "off hours" when there was relatively no network traffic (11 idle login accounts).

AVERAGE TIME (in seconds) FOR ONE CYCLE
(sending then receiving) FOR METHOD OF TRANSPORT

ARRAY CONFIGURATION	Iteration: one element at a time	Vector (RPC transport routine)
25 short ints	0.0098	0.0099
25 long ints	0.0101	0.0099
100 short ints	0.0125	0.0124
100 long ints	0.0129	0.0127
250 short ints	0.0170	0.0164
250 long ints	0.0171	0.0163
500 short ints	0.0237	0.0227
500 long ints	0.0248	0.0235

TEST #2

The second test was run during working hours when the network was processing an average of 51 active login accounts.

AVERAGE TIME (in seconds) FOR ONE CYCLE
(sending then receiving) FOR METHOD OF TRANSPORT

ARRAY CONFIGURATION	Iteration: one element at a time	Vector (RPC transport routine)
25 short ints	0.0106	0.0100
25 long ints	0.0105	0.0114
100 short ints	0.0128	0.0127
100 long ints	0.0127	0.0139
250 short ints	0.0171	0.0165
250 long ints	0.0174	0.0168
500 short ints	0.0244	0.0228
500 long ints	0.0248	0.0234

The activity on the network typically does slow down the time for one cycle, but the slow down is usually less than one millisecond. This is seen by comparing the time differences between TEST 1 (the off hours test) and TEST 2 (the working hours test).

The high-level RPC vector transport routine typically makes a slight time improvement over the more basic method of sending the array as a series of single integers. This difference is also usually less than one millisecond.

The time difference between sending shorts vs. longs is again usually less than one millisecond, with sending shorts typically the faster of the two. This fact may demonstrate that the true cost of the communication is in the client and server code that manages the sending and receiving.

CONCLUSIONS

The amount of data sent (arrays of 25 to 500 integers) is in the range of the amount of communication data anticipated for a distributed version of the PDK. From these data we see that a reasonable minimal estimate of the network communication overhead time of one remote call (C, in the previous section's notation) is about 0.01 seconds. This is the value that will be used in estimating possible speedups due to the proposed parallel implementation.

The differences between the various methods (busy vs. not busy network, iteration vs. vector transport, and long vs. short integers) are typically less than one millisecond. Since this is about an order of magnitude less than the communication overhead value itself, these turn out to be less of an issue for the speedup estimate. Nevertheless, the fastest method available will be used in any eventual implementation.

3.1.2.4. WHEN DOES PARALLEL PROCESSING BECOME WORTHWHILE?

Introduction

Several studies must be brought together to decide when a parallel implementation becomes worthwhile. An implementation plan has been given for parallelizing the most recent version of the re-engineered phonetic decoder kernel (PDK). This has been studied to estimate the algorithmic speedup available from a parallel implementation. The network communication overhead has been measured for the network currently available as a target architecture.

In this document the measurements of the current algorithm for prediction of the performance of the parallel implementation are completed. These studies are then put together to form specific predictions of parallel performance.

Measurements of the Sequential Algorithm's Parts

The decoder is divided into three conceptually separate parts. These parts correspond to the different ways that the code must be treated in a parallel implementation. One part is not parallelizable at all; it can only run sequentially. Another part is fully parallelizable; it can be run on as many processors as are available, up to its granularity limit. A final part must

be duplicated when executing in a parallel implementation; some effort that was performed in the sub-processes will be repeated in the main calling process. These distinctions were explained in the earlier section on algorithmic speedup estimations.

The code profiler gives statistics of the time spent in all the routines of a given program. The profiler was used to estimate the time spent in each of the parts of the decoder, to estimate the execution time of the parallel implementation. The decoder's subroutines were classified into the various parts of the algorithm as follows:

Parallelizable part:	Expansion and scoring of phones
Duplicated part:	Segment beam element duplication removal (uniquing)
Unparallelizable part:	Everything else

From earlier studies, it was known that the bulk of the execution time was being spent in the parallelizable part, so it was reasonable to include everything else" in the unparallelizable part. Further studies may show that other parts are indeed parallelizable. The parallelizable part was chosen for execution in parallel primarily because it is where most of the execution time is spent. As such, it has also been the subject of most of the optimization improvements to the code, as well as some of the re-engineering effort in preparation for parallelism.

The execution time results of the code profiler were as follows:

Parallelizable part:	60%
Duplicated part:	25%
Unparallelizable part:	15%

If we let:

p -- the fraction of time spent in the fully parallelizable portion
d -- the fraction of time spent in the duplicated in parallel portion
u -- the fraction of time spent in the unparallelizable portion

then these values are p, d, and u, respectively.

These percentage figures are out of a total average time of 1.66 seconds of SUN SPARCstation1 execution time per utterance. The average length of an utterance in this test case was 74.2 segments. Thus the average time to process one segment is about 0.02 seconds per segment. These averages are over 1025 utterances of a difficult generic test grammar, the performance measurement "pmtext" grammar.

The average time depends on the application complexity; therefore, this breakdown in time may be different for different applications. However, the distribution of time spent in the various parts of the code will probably be similar to these results (at least roughly) for different applications. Thus these percentages will be used in the following speedup estimates.

Speedup Predictions

First, we review the definitions of the following variables and formulas:

U -- the unparallelizable portion's execution time per segment
P -- the fully parallelizable portion's execution time per segment
D -- the duplicated in parallel portion's execution time per segment

C -- the network communication overhead time per call
N -- the number of available sub-processors, including main routine's machine

T -- the total time of execution per segment

Our expression for the sequential decoder's execution time per segment:

$$T_s = U + D + P$$

Our expression for the parallel decoder's execution time per segment:

$$T_p = U + D + D/N + P/N + C*N$$

Strictly speaking, the parallel version will give a speedup whenever T (parallel) is less than T (sequential). This is satisfied whenever:

$$T_p = (u + d)*T_s + (d + p)*T_s / N + C*N < T_s$$

We will evaluate this expression for the values found for our current "home" network computing environment to see if there is anything to be gained from the proposed parallel implementation in this environment. Other environments can be evaluated in a similar manner, after the appropriate values are measured.

From the expression of the sequential decoder's execution time, we can get the values for U, D, and P, from the percentage of the time spent in each of these portions of the code:

$$T_s = U + D + P = uT_s + dT_s + pT_s$$

$$T_s = 0.02; \quad U = T_s * .15, \; D = T_s * .25, \; P = T_s * .6;$$

U = 0.003, D = 0.005, P = 0.012 (all in seconds per segment).

From the previous section estimating the network communication overhead, we have a value for C, which is about 0.01 seconds. Now we have all the terms needed for evaluating the expression for the parallel decoder's execution time:

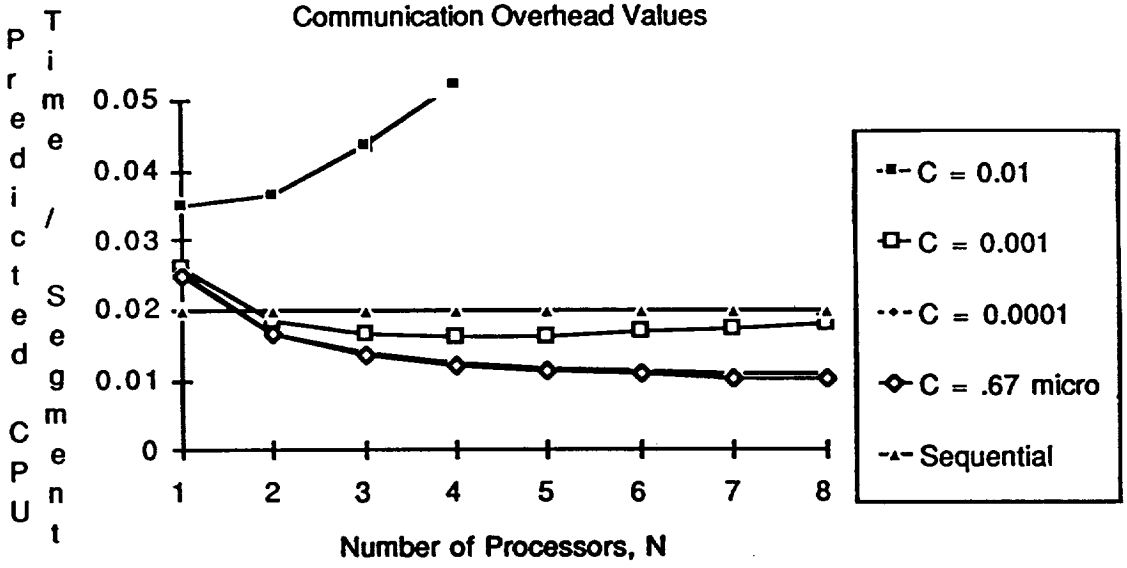
$$T_p = 0.003 + 0.005 + 0.005/N + 0.012/N + 0.01*N,$$

$$T_p = 0.008 + 0.017/N + 0.01*N$$

From this final formula, we can see that the linear in N communication overhead term dominates right away. When N = 2, the value for T (parallel) 0.0365 is already greater than the original value for T (sequential) =.02. The value for C in the case of our SUN network is too high for us to reap any benefit from the proposed parallel implementation.

The parallel execution times per segment for several values of N for this configuration are plotted in figure 3.1.2.4.1. Also included in this figure are the values for C = 0.001, a communication overhead only one order of magnitude less than that actually measured. This shows that for even a relatively small improvement in the communication overhead the parallel implementation will improve the speed of decoding by up to 20%, as in this example for N = 4. Values for C = 0.0001 show a potential speedup of 40% at N = 5.

Figure 3.1.2.4.1. Prediction of CPU Time per Segment for Various Communication Overhead Values



Of course smaller communication overhead values would produce even greater speedups. A more tightly coupled group of processors would naturally have a smaller communication overhead time. As a sort of limit to what the minimum overhead of a tightly coupled system might be, the average time of a subroutine call was measured for the SUN SPARCstation1. A program was written that measured the time to call two subroutines (to model the communication method proposed in the earlier section). It measured the time of 5 million such calling cycles. The average time for a call was about 0.67 microseconds. Figure 1 also includes a plot using this value for C. This shows the potential for decoder acceleration by running on a very tightly coupled set of parallel processors.

Observations

The estimate of the parallel implementation's execution time has three terms, a constant, a linear term, and a 1/N term. The coefficients of these terms will vary according to the measured values for any particular target environment. Still, we can conclude certain things from the form of the estimate alone.

As N increases, eventually the linear term will dominate. If C is sufficiently small, there is a range of N for which T (parallel) will be less than T (sequential). In this range, T (parallel) will be decreasing until the linear term's increase balances the 1/N term's reduction. This will produce an optimum point in the time reduction range. The smaller C is, the larger this range is. For C small enough, T (parallel) is still decreasing when the parallelism granularity limit on N is reached. This situation would provide the most benefit from a parallel implementation, where it is always beneficial to use more processors.

Accuracy Improvements

Once a parallel implementation is available that really does decrease the decoding time, we may want to use the extra resources for accuracy improvement rather than for speed improvement. The parallel decoder may be reconfigured to do a deeper search in about the same execution time as the original sequential implementation. This would be done by increasing the segment beamwidths so that the total parallel time is the same as the original sequential execution time of the shallower search. Any tradeoffs along these lines would be evaluated when selecting the search constraint parameter settings for the parallel implementation.

Conclusions

An expression has been given for determining when the proposed parallel implementation will be faster than the current sequential implementation of the Phonetic Decoder Kernel. This expression requires estimates of the sequential decoding time per segment, the relative time spent in each of the different parts of the code (unparallelizable, duplicated in parallel, and fully parallelizable), and the communication overhead.

For our "home" environment of SUN SPARCstation1 workstations communicating via an Ethernet network, the communication overhead is too large to see any improvement in time from the proposed parallel implementation. An important reason why the parallel implementation will not produce a time improvement is that the time has already been substantially reduced during the research phase preparing for a parallel implementation. Still, an improvement would have been seen if the communication overhead had been an order of magnitude smaller, which should not be much considering our current network is very loosely coupled. As other parallel hardware becomes available, the possible decoding time improvement of porting to a parallel implementation should be reevaluated.

3.1.3. PARALLEL PROCESSING FOR SPEECH RECOGNITION: REVIEW OF RELATED WORK

A good description of SSI's Phonetic Decoder prior to the start of the parallel processing project can be found in [Meisel 88]. This describes the basic two-level (word and segment) approach, with none of its improvements for parallel processing. The basic one-level, one-pass beam search is described in [Lowerre 80]. A comparison of several types of decoding

algorithms is given in [Godin 89]. This compares a one-level, a two-level and a level-building decoding approach. The two-level approach described there is not quite like SSI's initial two-level approach, and should not be confused with it.

The parallel word scoring and unique word scoring projects are unlike anything we have found in the literature, probably because they are relatively specific to SSI's two-level approach. These projects have lead us to the eventual implementation of a single-level approach, for which they have motivated similar projects for the new algorithm.

Most systems, e.g. [Bahl 83] and [Lee 88], have components for relative thresholding and coarticulation. By including relative thresholding they also have dynamic beam sizes, at least implicitly. These areas are different in our implementation because they have been implemented with a view toward an eventual parallel implementation.

SSI's implementation of a single-level beam search decoder is different from the standard version of this algorithm, primarily because it still has parts of the old two-level version. These were kept as a natural way to save and recover the decoded word sequence, and to organize considerations for using a grammar and/or language model.

Other systems have implemented a parallel version of their algorithm and achieved a significant speedup. (See the report in [Kumar 86], for example.) Another approach is to use custom hardware designed with the speech recognition task in mind (e.g., [Bisiani 89]). Yet another approach is to use parallel speech recognizers, as in [Stolfo 89].

Work continues on the related effort of parallelizing the underlying dynamic programming algorithm of many speech recognition systems, as reported on in [Charot 86] and [Stainhaouer 90]. These results should prove very interesting as parallel hardware becomes more generally available.

SSI's primary computing environments have supported a degree of parallelism, initially with the VAX-8800 [DEC 86a, 86b] and currently with the SUN network [SUN 88]. Improvements to the communication overhead of the SUN network are necessary before we can gain any speed increase from changing to a parallel implementation.

Still, there seems to be less interest in parallel implementations of speech recognition algorithms recently, considering the dearth of papers dealing with parallel implementations in the speech sections of the recent International Conference on Acoustics, Speech, and Signal Processing (1990). We suspect that the availability of inexpensive, fast, general-purpose computing environments (combined with improved algorithms) is leading other researchers to a similar conclusion to ours: very good time-accuracy tradeoff points for speech recognition can be found on sequential machines. This makes parallel implementations less appealing if there already exists an established base of potential customers with sequential machines.

3.1.4. PARALLEL PROCESSING IN THE PHONETIC DECODER: SUMMARY AND CONCLUSIONS

There are two major parts to the parallel processing in the Phonetic Decoder project: the "Decoder Re-engineering for Parallelism" part and the "Parallel Processing Predictions"

part. Both parts have been successful, and have improved our understanding of how to make a better decoder.

The re-engineering part has greatly modified the decoder in preparation for a parallel implementation. This re-engineering took the form of several sub-projects: parallel word scoring, dynamic beam sizes, relative thresholding, coarticualtion simplifications, unique word scoring, and single-level beam search. These changes were made primarily for the parallel implementation we were planning to evaluate; however, all of these changes produced improvements in the sequential implementation. The re-engineering part has significantly improved the decoder's performance, by about a 1-5% improvement in absolute utterance accuracy and by about a 75-85% decrease in decoding time.

As the re-engineering work proceeded, our idea of what the appropriate parallel implementation should be also changed. The prediction part of the project made predictions of the parallel implementation suggested by the results of the re-engineering part.

The parallel processing prediction part of the project has estimated the changes to the execution time of decoding as a result of a possible parallel implementation. This part has derived an expression for estimating when the proposed parallel implementation becomes worthwhile. The parameters of this expression were estimated for SSI's current "home" computing environment, a network of SUN workstations. It was found that the communication overhead of this environment is too slow to produce any speedup in the decoder from parallelism, and yet a comparatively small improvement in the communication overhead would produce decoding time improvements in the parallel implementation.

The speed improvements resulting from this project go a long way in making SSI's speech recognition system more acceptable to its eventual users. Since there have been accuracy improvements along with the speed improvements, the speed improvements have not been gained with a loss of accuracy. The speed improvements are available in the sequential version of the Phonetic Decoder, so special purpose hardware is not necessary for these speed improvements. The availability of inexpensive and fast workstations has somewhat eased the problem of decoding time as well.

A parallel implementation of the Phonetic Decoder was originally proposed for this project as a method of improving system response time. Although the parallel version was never implemented, we now know how to evaluate a computing environment to determine when the parallel version becomes worthwhile. The speed improvement present in the sequential version of the decoder has brought the response time down to a reasonable level. The goals of this project have thus been reached.

3.2. NOISE SENSITIVITY REDUCTION

3.2.1. NOISE IMPACT STUDY

Abstract

This task involved studying the impact of noise on the recognition accuracy. Several experiments were performed to evaluate the effect of different levels and two types of noise on the recognition performance. We also examined profiling as a means of dealing with noisy environments. We will present results that show high levels of noise in the speech data adversely affect the recognition accuracy performance. At the same time, we will see

how profiling a speech recognition model on noisy data can improve its performance under noisy conditions.

In section 1, we will discuss our approach in conducting this study. Section 2 contains an overview of the results. In section 3, we present a detailed analysis of the change in the behavior of the system resulting from the presence of noise in the speech data. The detailed results will be presented in section 4. Finally, section 5 contains a few general conclusions.

3.2.1.1. Introduction

We used two different types of noise for our studies. A random number generator with a normal distribution of mean zero was used to create a *white* noise signal. We also collected a digitized audio signal file containing the environment noise in one of our offices. The main source of noise, in this case, was a SONY optical disk drive. The plot of this noise signal's magnitude spectrum shows that it is very pinkish. While *white* noise has equal power in different bandwidths (flat spectrum), *pink* noise is not nearly as powerful in the high-frequency range.

We used the two types of noise signal to generate different levels of noisy data. This was performed by adding the noise signals to the digitized speech data (collected from a single speaker) while trying to achieve a desired signal-to-noise ratio (SNR) and keeping the RMS of the resulting noisy speech signal equal to that of the original speech signal. While this method of generating noisy data has the advantage of allowing us to use the same speech at different noise levels, we know that people tend to speak differently in the presence of significant amounts of background noise. Therefore, the impact of environment noise on a speaker's speech characteristics is not evaluated here. In this report, *clean* speech refers to the data which contain *no added* noise. Also, we used constant-frame-rate acoustic processing to convert the digitized audio data for each utterance to a sequence of 6.6 millisecond frames of speech.

Very early in our investigations, we discovered certain limitations in the phonetic decoder with respect to the decoding of the silence word. A silence word, which should also be viewed as a non-speech or noise word, is decoded to deal with the presence of silence or noise in the speech data. By removing the silence-word related constraints from the decoder, we were able to achieve significant accuracy improvements on data with high levels of noise in many cases and the system became more robust with respect to the background noise. The results reported here were obtained using this new version of the decoder.

3.2.1.2. Overview of the Results

High levels of background noise result in noticeable recognition accuracy degradations. However, the recognition performance in environments with high noise levels can be significantly improved by profiling a generic speaker model (trained on clean data) on noisy speech data. There are several observations that can be made based on the data presented in section 4.

White Noise vs. Pink Noise:
While relatively low levels of white noise result in significant accuracy degradations, higher levels of pink background noise are needed to see noticeable drops in the accuracy. The accuracy of the generic speaker model is worse on the white noisy data (at 30dB

signal-to-noise ratio) than on the data containing pink noise (at 25dB). Using the two data sets for profiling results in models with a smaller gap in their performances on their respective data sets (30dB profiled model on 30dB data vs. 25dB profiled model on 25dB data).

To make the following observations regarding the impact of noise, we focused our attention on those test cases which involved the use of data containing pink environment noise (more realistic scenarios than white-noise test cases). The graph at the end of this section (figure 3.2.1.1) depicts some of the results.

- Using the clean data for profiling not only improves the recognition accuracy on the clean data, but also results in better performance on the noisy data.

- Profiling the generic model on the noisy data (signal-to-noise ratio of 25dB) results in a model with improved recognition accuracy on data with different levels of noise. The resulting model also yields higher accuracy (than the generic model) on the clean data.

- The best performance results are achieved when the generic speaker model profiled on clean data is evaluated on the clean test data.

- At very low noise levels, test data with 40dB signal-to-noise ratio, the profiled model created using clean profiling data outperforms the model generated using 25dB profiling data.

- At higher noise levels, test data with 30dB signal-to-noise ratio, the profiled model created using clean profiling data performs slightly better than the model generated using 25dB data.

- At medium noise levels, test data with 25dB signal-to-noise ratio, the profiled model created using the 25dB profiling data set outperforms the profiled model generated using clean profiling data.

- Finally, at a high level of noise, test data with 15dB signal-to-noise ratio, the generic model profiled on 25dB data clearly outperforms the profiled model created using clean data.

3.2.1.3. Analysis

We can study the behavior of the generic speaker model (trained on clean speech data) on data sets containing different levels of noise. Note the relationship between the changes in the recognition accuracy and the average number of segments per utterance (output by the phonetic encoder). At low levels, noise shows up in the encoded data as spurious segments of the noise segmentation class (which also contains the release portions of stops) within silence (including stop part of stops) portions of each utterance. As the noise level increases, the number of noise segments increase. At the same time some of the noise segments become longer; i.e. more dominant within silence. At high levels, noise completely dominates the silence portions of each utterance and we see a drop in the average number of segments. Sequences of <Stop,StopRelease> segments are reduced to single noise segments throughout the utterances. While the above effects are the most noticeable ones, the presence of noise in the data also impacts the encoding of the speech portions of each utterance. The extent of this impact is obviously a function of the level and the type of the noise. The behavior of the model changes once it is profiled with noisy data.

Higher levels of pink background noise have a less devastating impact on the recognition accuracy than some lower levels of white noise. White noise is more powerful in the high-frequency range. Medium to high levels of white noise result in the frequent formations of spurious (and dominant) runs of the strong-fricative segmentation class.

3.2.1.4. Results

Speaker: NASA female speaker JB (with a southern accent)

Data Sets:

D0: Clean speech data (no noise added, estimated SNR of about 45dB).

D40P: 40dB data (environment noise added)
D30P: 30dB data (environment noise added)
D25P: 25dB data (environment noise added)
D15P: 15dB data (environment noise added)

D40W: 40dB data (white noise added)
D30W: 30dB data (white noise added)
D20W: 20dB data (white noise added)

Data sets D0, D25P, and D30W contain 950 profiling and 198 testing utterances. The other sets contain only 198 testing utterances.

Models:

M0: 3008, 3.3 general release generic female model
M0_0: M0 profiled on the profiling utterances from D0
M0_25P: M0 profiled on the profiling utterances from D25P
M0_30W: M0 profiled on the profiling utterances from D30W

SliderSetting: 6

Syntax: nasaonly.nec (a NASA test application)

All the recognition tests were performed on a SUN Sparcstation1.

I) RECOGNITION PERFORMANCE RESULTS OF DIFFERENT MODELS ON EACH DATA SET.

Data Set: D0

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	90.56	91.18	74.24	81.82	91.41	42.87	0.47
M0_0	95.45	95.53	81.31	93.43	94.95	43.95	0.89
M0_25P	93.18	92.32	76.26	85.86	92.42	54.68	1.16
M0_30W	93.26	93.97	76.26	89.39	95.45	49.54	0.87

Data Set: D40P

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	91.24	92.33	76.26	84.85	92.93	43.80	0.49
M0_0	95.70	96.60	83.33	92.93	96.97	45.10	0.92
M0_25P	94.52	93.81	78.79	89.90	93.43	59.74	1.34

Data Set: D30P

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	90.40	91.40	70.71	80.81	92.42	54.04	0.86
M0_0	93.85	94.57	77.78	88.89	94.95	55.45	1.44
M0_25P	93.18	92.47	72.73	88.89	93.94	49.11	1.09

Data Set: D25P

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	80.79	84.20	58.59	70.71	83.33	55.67	1.02
M0_0	88.12	90.41	68.69	81.82	89.90	57.21	1.62
M0_25P	92.75	92.75	74.75	87.37	92.93	59.24	1.32

Data Set: D15P

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	41.87	50.61	16.16	30.30	43.43	32.47	0.55
M0_0	60.15	66.42	32.32	47.47	58.08	35.03	0.86
M0_25P	78.26	76.78	49.49	64.65	71.72	62.68	1.69

Data Set: D40W

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	85.59	87.06	63.13	78.79	85.86	54.96	0.97
M0_0	90.99	92.07	68.69	85.35	92.93	56.10	1.50
M0_30W	86.18	87.36	67.17	79.80	87.88	60.68	1.43

Data Set: D30W

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	53.83	60.11	25.25	40.91	50.00	66.08	1.12
M0_0	46.17	51.02	22.22	37.37	50.00	68.64	1.75
M0_30W	88.21	89.64	63.64	77.27	90.40	44.55	1.03

Data Set: D20W

Model	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
M0	11.88	15.13	0.51	6.06	13.64	46.80	0.49
M0_0	15.59	22.21	3.54	8.08	17.68	41.38	0.70
M0_30W	16.34	19.94	3.54	7.07	15.15	67.19	0.98

II) RECOGNITION PERFORMANCE RESULTS OF EACH MODEL ON DIFFERENT DATA SETS

Model: M0

DataSet	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
D0	90.56	91.18	74.24	81.82	91.41	42.87	0.47
D40P	91.24	92.33	76.26	84.85	92.93	43.80	0.49
D30P	90.40	91.40	70.71	80.81	92.42	54.04	0.86
D25P	80.79	84.20	58.59	70.71	83.33	55.67	1.02
D15P	41.87	50.61	16.16	30.30	43.43	32.47	0.55
D40W	85.59	87.06	63.13	78.79	85.86	54.96	0.97
D30W	53.83	60.11	25.25	40.91	50.00	66.08	1.12
D20W	11.88	15.13	0.51	6.06	13.64	46.80	0.49

Model: M0_0

DataSet	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
D0	95.45	95.53	81.31	93.43	94.95	43.95	0.89
D40P	95.70	96.60	83.33	92.93	96.97	45.10	0.92
D30P	93.85	94.57	77.78	88.89	94.95	55.45	1.44
D25P	88.12	90.41	68.69	81.82	89.90	57.21	1.62
D15P	60.15	66.42	32.32	47.47	58.08	35.03	0.86
D40W	90.99	92.07	68.69	85.35	92.93	56.10	1.50
D30W	46.17	51.02	22.22	37.37	50.00	68.64	1.75
D20W	15.59	22.21	3.54	8.08	17.68	41.38	0.70

Model: M0_25P

DataSet	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
D0	93.18	92.32	76.26	85.86	92.42	54.68	1.16
D40P	94.52	93.81	78.79	89.90	93.43	59.74	1.34
D30P	93.18	92.47	72.73	88.89	93.94	49.11	1.09
D25P	92.75	92.75	74.75	87.37	92.93	59.24	1.32
D15P	78.26	76.78	49.49	64.65	71.72	62.68	1.69

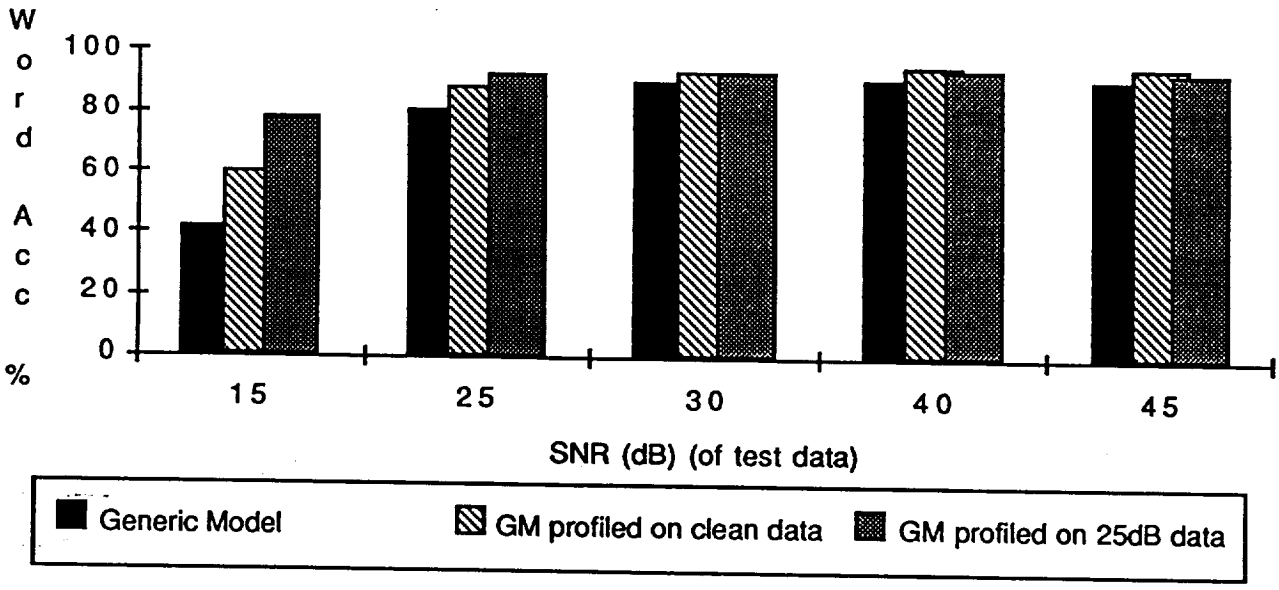
Model: M0_30W

DataSet	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
D0	93.26	93.97	76.26	89.39	95.45	49.54	0.87
D40W	86.18	87.36	67.17	79.80	87.88	60.68	1.43
D30W	88.21	89.64	63.64	77.27	90.40	44.55	1.03
D20W	16.34	19.94	3.54	7.07	15.15	67.19	0.98

5. Conclusion

The recognition accuracy of our system is fairly robust with respect to low to medium levels of environment noise. Higher levels of noise, 25dB signal-to-noise ratio for example, can result in significant accuracy degradations. However, profiling (with noisy data) can be used to recover most of the lost accuracy under noisy conditions.

Figure 3.2.1.1. Environment Noise and Recognition Accuracy



3.2.2. CONSTANT FRAME RATE MODELS

This section gives the results of building models with a constant framing rate in the acoustic processor. A new constant frame rate model is compared with the corresponding pitch synchronous model. Building models with a constant framing rate is a first step in preparing to study some of the alternate acoustic processing methods, as they work only on constant frame rate data. Constant frame rate models also add some noise tolerance to models, since the model is no longer required to track the pitch (which may be affected by certain types of noise). A constant frame rate acoustic processor also makes models more

speaker independent (as long as they all use the same constant rate), improving both generic and profiled models.

COMPARISON OF CONSTANT FRAME RATE AND PITCH SYNCHRONOUS MODELS

OVERVIEW

In this report we compare the performance of models generated during some of the intermediate steps of model building, as well as running independent speaker tests on two different three-speaker generic male (3GM) models: 1215, the "baseline" Pitch Synchronous Model (PS) and 1279, the Constant Frame Rate Model (CFR). The speakers for building and testing both models are the same. The results show that both models achieve very similar performance.

MODEL BUILDING METHODS

Three male speakers were chosen for the new CFR model, each with about 830 utterances for model building, for total of 2489 model build utterances. The details of the two model builds are given in the following table:

	Pitch Synchronous	Constant Frame Rate
Model Number	1215	1279
Type	3GM	3GM
Bootstrap prf	1004	1004 (release 3.1 generic male)
Input files	PS (regular)	CFR (smoothed)
Model Build (MB) Version	3.2	3.2 +
Adapt & MB / PDK Version	3.0	3.0
Grammar	pmtext.nec	pmtext.nec
Ind. Tests / PDK Version	3.1	3.1
Grammar(s)	pmtext.nec nums_infinite.nec	pmtext.nec nums_infinite.nec
Sun(s) used/ Adapt & MB	various	various
Ind. Tests	sun-4/280	sun-4/280

Note that the Constant Frame Rate model had its input files smoothed before model building. Both models used the same bootstrap which is pitch synchronous. With the exception of input files being processed by constant frame rate, being smoothed, and a slight difference in the model build version, the model building methods were identical for both models. For a complete comparison of Pitch Synchronous and Constant Frame Rate models, we would need to see the performance of a CFR model with non-smoothed input files as well as a smoothed PS model. (The smoothing method is described in the next section.)

DEPENDENT SPEAKER TESTS

These are the average scores of the three dependent speakers for both models during different stages of model building. Since the model building was made on different machines over a span of weeks, the Ave Time cannot be used in any comparisons. The completed (postbuild) models are nearly identical in dependent speaker performance.

utterance type: performance measurement
number of utts: about 130 per speaker, 384 total
grammar: pmtext.nec
PDK version: 3.0, 27-MAR-89

model: 1215 (PS)

stage	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
preadapt	63.21	64.23	18.72	43.85	61.03	64.04	17.25
postadapt	74.28	75.28	22.31	54.36	72.31	64.72	19.83
postbuild	81.64	84.15	31.79	66.41	83.08	66.51	15.15

model: 1279 (CFR)

stage	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
preadapt	70.86	71.48	17.97	47.92	66.15	71.48	22.35
postadapt	75.83	75.52	21.35	56.25	71.61	71.56	24.66
postbuild	81.88	83.99	31.77	65.10	84.11	63.16	13.52

REMARKS ON DEPENDENT SPEAKER RESULTS

In the results for dependent speakers, a very noticeable feature is that the CFR model has a higher Word Acc and W Trans that the PS model at the preadapt stage. This is peculiar since in preadapt nothing has happened to the bootstrap model as yet, so we're seeing a PS bootstrap get better performance on CFR smoothed input files than on PS input files. At postbuild, the Word Acc and Utt Acc are amazingly close. This tells us that the CFR model is perhaps not better than the PS model, but it also isn't worse. Nothing was lost by making a model with CFR input files.

INDEPENDENT SPEAKER TESTS

Five independent male speakers were used for the pmtext testing. None of these speakers contributed to either model. Here are the average scores for all five speakers. The CFR model is significantly faster than the PS model (the independent speaker test times are comparable).

utterance type: performance measurement
number of utts: 130 per speaker, 650 total
grammar: pmtext.nec
PDK version: 3.1, 15-AUG-89

model: 1215 (PS)

slider setting	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
1	59.26	59.57	16.15	37.38	54.46	69.83	5.00
4	73.19	73.13	22.00	48.15	68.31	70.38	18.51
7	75.74	75.15	23.23	50.92	70.00	70.31	26.22

model: 1279 (CFR)

slider setting	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
1	56.71	59.89	16.46	40.62	54.00	59.55	4.18
4	73.10	74.59	22.31	51.38	69.23	60.36	15.03
7	74.54	76.97	23.23	52.31	72.62	60.29	21.50

utterance type: digits
number of utts: about 200 per speaker, 999 total
grammar: nums_infinite.nec
PDK version: 3.1, 15-AUG-89

model: 1215 (PS)

slider setting	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
1	85.39	69.03	30.83	58.26	75.78	27.75	0.31
4	85.96	70.00	31.53	58.06	76.68	27.75	0.48
7	85.81	69.86	31.13	57.66	76.78	27.75	0.66

model: 1279 (CFR)

slider setting	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Segs/Utt	Ave Time
1	82.79	70.70	29.53	57.16	78.18	23.61	0.24
4	83.72	71.19	30.33	59.96	78.78	23.60	0.37
7	83.93	71.20	29.73	59.76	79.68	23.60	0.52

REMARKS ON INDEPENDENT SPEAKER RESULTS

Examining the results for tests using independent speakers, the CFR model would appear to be about 20% faster than the PS model for both kinds of tests, pmtext and digits. With pmtext, the Word and Utt Acc's are very close, and with digits the scores are slightly lower for the CFR model.

CONCLUSIONS

We conclude from these results that we can build constant frame rate models with similar performance to our current pitch synchronous models, with a possible improvement in decoding speed. This allows us to do a variety of tests with new acoustic processing methods that are suitable only for constant frame rate data. Now we know that these other

acoustic methods will get a fair trial, since the constant frame rate models can perform as well as the pitch synchronous models. Furthermore, using a constant frame rate may improve speaker independence in generic models.

3.2.3. SMOOTHING OF ACOUSTIC PROCESSOR OUTPUT

OVERVIEW

One of the simplest things to try when implementing a constant rate acoustic processor is a smoothing method on the parametric output of the acoustic processor. This is now easier to implement because the smoothing method can assume that the frames are of constant duration; no adjustment is required for frames of different duration.

Smoothing methods are typically used for improved noise robustness, since they remove small "glitches" in the data that show up for short durations. Smoothing for noise this early in the system's processing helps all of the following processes: the phonetic processor and the phonetic decoder.

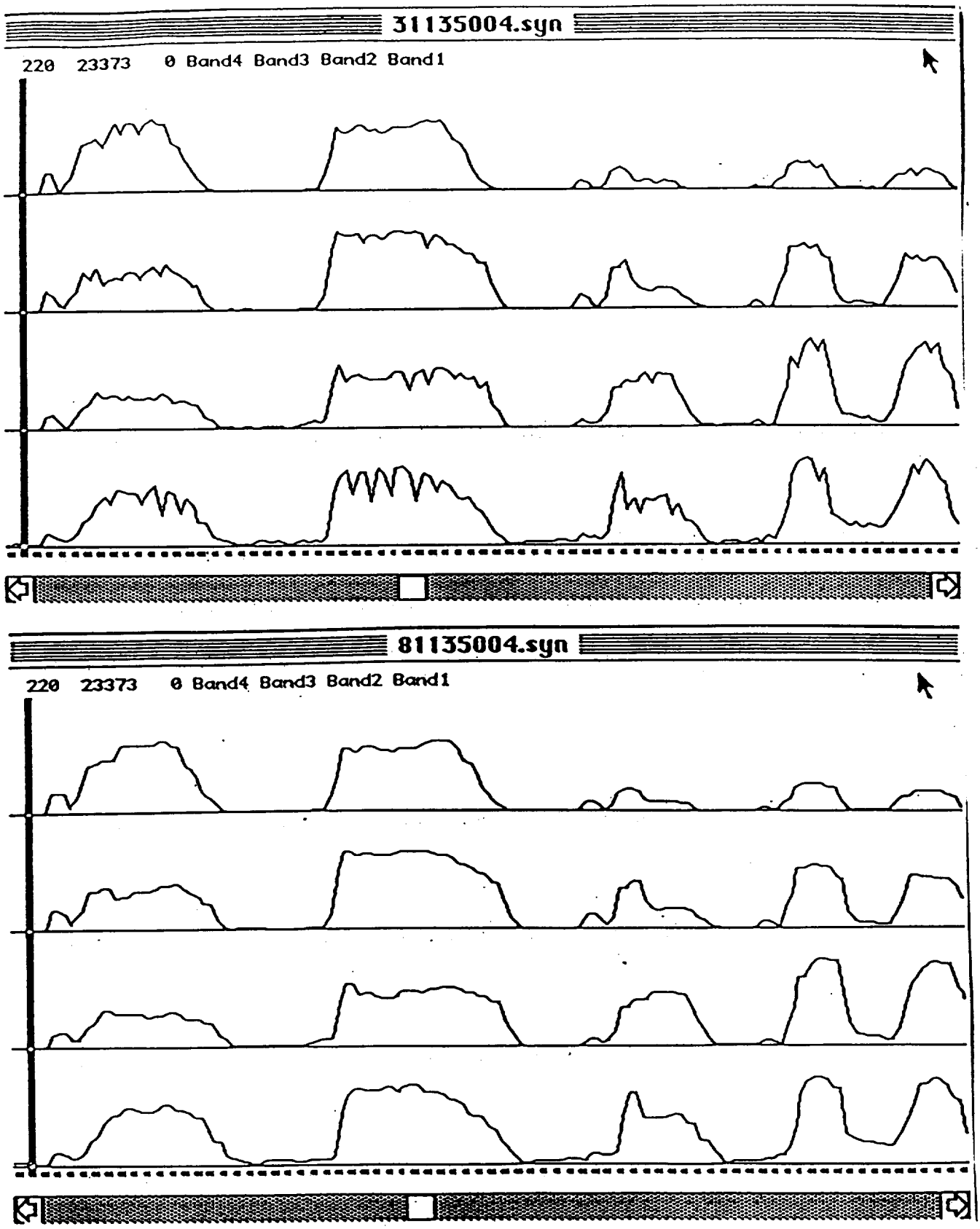
A smoothing method was tried, and its results are reported in this section. This method has become our default acoustic processing output smoothing method.

SMOOTHING METHOD

The smoothing method we have tried is a very simple one. It smooths a frame parameter by replacing the value of the frame parameter with the maximum of the current frame and the previous frame for: the frequency bands, the kinetic magnitude, and the magnitude parameters. The first frame of an utterance is left untouched.

Figure 3.2.3.1 gives a graphic example of the smoothing for several of the parameters involved. This shows the values for frequency bands one to four, for a portion of the sentence "Greek music also contributed the concept of intervals," the portion being "also contribut-." This figure clearly shows how the smoothing method removes the short dips in the signal, and better captures the "envelope" of the speech data.

Figure 3.2.3.1. Unsmoothed (top) vs. smoothed (bottom) acoustic parameters.



TEST RESULTS

Tests were run doing only the smoothing of the test input parameter files. This means that the phonetic speech models were not rebuilt with the smoothed data, thus somewhat penalizing the potential of the smoothed method. If the results are better for the smoothed version using only this change, we can expect the results to be better still when the full speech model is built on the smoothed data.

Three speakers supplied test utterances for a generic six-speaker model, number 1213. The 3.1 PDK version was used (890815), and all the tests occurred in near identical conditions so that the times are comparable.

The three speakers are :

PS 1219
WSM 1220
LSO 1224

The results of these tests are given in the tables below. These results compare the smoothed and unsmoothed versions. The smoothed versions are compared at two different slider settings, indicated with "ss." The results are given for each speaker. The average across all speakers is also given. The grammar used here was the pmtext test application.

1219	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
unsmoothed/ss4	81.76	81.47	31.54	65.38	84.62	29.25
smoothed /ss4	81.97	81.45	26.15	58.46	75.38	21.94
smoothed /ss7	83.39	82.98	29.23	60.00	81.54	29.38

1220	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
unsmoothed/ss4	76.58	77.29	26.15	56.15	73.08	29.09
smoothed /ss4	80.27	80.90	23.08	59.23	80.00	22.43
smoothed /ss7	82.40	82.40	26.15	60.77	83.85	30.32

1224	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
unsmoothed/ss4	73.88	76.66	17.69	46.15	69.23	32.64
smoothed /ss4	67.28	68.30	20.77	40.00	56.15	21.40
smoothed /ss7	67.57	68.19	22.31	38.46	59.23	29.34

average	Word Acc	W Trans	Utt Acc	+ Off 1	+ Off 2	Ave Time
unsmoothed/ss4	77.41	78.47	25.13	55.89	75.64	30.33
smoothed /ss4	76.51	76.88	23.33	52.56	70.51	21.92
smoothed /ss7	77.79	77.86	25.90	53.08	74.87	29.68

The initial smoothed test, at slider setting 4, produced results that were sometimes more accurate and sometimes less accurate, but were always faster. To evaluate further the potential of the smoothed method, tests were run at slider setting 7. At the higher/slower slider setting, the accuracy of the smoothed method is usually better than that of the unsmoothed method, while still being faster than the unsmoothed method at slider setting 4.

This shows that slider settings for the smoothed method can be found that produce accuracy improvements for a given equivalent decoding time. Thus, although somewhat hidden by the current choice slider settings, the smoothed method can be tuned to produced a better overall system.

CONCLUSIONS

A simple smoothing method has been implemented that improves system performance. This method will be included in subsequent model building, and will require re-tuning of the slider setting parameters.

3.2.4. ALTERNATE ACOUSTIC PARAMETERIZATION

Abstract

This report describes a study of an alternate representation scheme for the acoustic information used in speech recognition, and its comparison with the existing method currently in use at SSI. Both schemes use fixed-duration time intervals (frames) as speech acoustic units. In both schemes, frames are represented as vectors of acoustic parameters.

In the first scheme, which is currently employed at SSI, the parameters are computed using the amplitudes output by bandpass filters of a filterbank, and an estimated pitch period. It is called here "FBP AP" (FilterBank and Pitch Acoustic Parameterization).

As an alternate scheme, we selected a widely-used method of the acoustic frame parameterization utilizing mel-scale cepstrum coefficients and power. It is called here "CEP AP" (Cepstrum and Power Acoustic Parameterization).

Two schemes of acoustic parameterization are compared by applying them to a speaker-independent large vocabulary continuous speech recognition task. The DARPA Resource Management (RM) task (a well-established benchmark for DARPA speech recognition program) was used in the experiments. Two speech recognition models (one for each of the above acoustic parameterization schemes) were built and evaluated using the RM training and test speech data. The results show that the existing method (using filterbank output and pitch period estimate) is comparable to the alternative method (using cepstrum coefficients and power) in terms of recognition performance.

Introduction

The acoustic processing scheme employed in the SSI speech recognition system uses a filterbank analyzer and a pitch tracker to perform acoustic parameterization of fixed-duration frames. Studies of alternative acoustic parameterizations [Davis and Mermelstein 80] suggest that centisecond mel-cepstrum coefficients (computed over a 20-msec Hamming window) can be successfully used as frame parameters representing acoustic input to a speech recognition system. This frame parameterization is used in many of the existing HMM-based speech recognition systems (e.g., with discrete density HMMs [Lee 89], continuous density HMMs [Russell *et al.* 90], and mixed density HMMs [LeeCH *et al.* 90, Paul 89]).

Comparisons of efficiencies of the filterbank and cepstrum parameterizations applied to speech recognition tasks indicate possible superiority of the latter. For example, the ARM

system [Russell *et al.* 90] showed a significant performance improvement when a filterbank-based frame parameterization was replaced with the cepstrum-based one. In view of the this result (as well as similar ones reported in the literature), and the widespread use of the cepstrum-based parameterization, we selected it for testing as an alternate representation of the acoustic input for the SSI speech recognizer.

3.2.4.1. Acoustic Processing

Prior to application of any of the two schemes, speech waveform is sampled at 16 kHz and pre-emphasized with a filter of 1-0.97/z. The sample values are quantized into 14-bit integers. Each of the acoustic processing schemes is applied to the same stream of speech waveform samples. Below we describe both schemes in some detail.

3.2.4.1.1. FBP AP

A gain control algorithm is applied to a stream of speech waveform samples, attempting to keep their values within a pre-defined dynamic range. A 4th-order FIR filter is applied to resulting stream of samples to further flatten the long-time spectrum of speech waveform (some flattening is done by the pre-emphasis filter). Following that, a filterbank analysis is performed. The filterbank is comprised of 20 2nd-order IIR bandpass filters. The set of 20 frequency passbands of the component filters are equally spaced on the Bark scale [Zwicker 61]. Each filter has a bandwidth of 1 Bark. Total band covered by the filterbank is approximately 2 Hz to 8 kHz.

A stream of samples output by each component filter is blocked into consecutive 10-msec time intervals (frames), and the maximum of the absolute values of the output samples within the frame is assigned as an acoustic parameter. Thus, 20 acoustic parameters, each of which represents a frame-wide peak of the absolute values of amplitudes output by a bandpass filter, are assigned to the corresponding frame. Three time-domain parameters are also computed for the time interval covered by a frame. These are frame-wide peaks of absolute values of waveform samples, and those of the time-differenced sample values, respectively; and an estimated pitch period which overlaps with the frame. Pitch period is computed by a pitch-tracking algorithm (for voiceless sounds, the pitch value is not defined, and has to be "faked").

An acoustic processing scheme close to the one described above is currently in use at SSI [Meisel *et al.* 89].

In summary, FBP AP's frame is a 10 msec time interval characterized with 23 parameters: (a) peaks of the absolute values of 20 bark-scale bands' amplitudes, (b) peaks of the absolute values of waveform sample magnitude, and those of the time-differenced sample magnitude, and (c) an estimate of the pitch period overlapping with the frame.

3.2.4.1.2. CEP AP

As in the FBP AP scheme, a stream of 16-kHz samples is blocked into frames. Each frame spans 20 msec; consecutive frames overlap by 10 msec. Each frame is multiplied by a Hamming window with a width of 20 msec and applied every 10 msec. We can redefine frames as non-overlapping 10-msec time intervals, which use a "look-ahead" window of 20 msec to access speech samples.

From the Hamming-windowed speech waveform samples, we compute the LPC coefficients using the autocorrelation method [Markel & Gray 76]. LPC analysis is performed with order of 14. Next, a set of 12 LPC-derived cepstrum coefficients is

computed from the LPC coefficients. These cepstrum coefficients are transformed to a mel-frequency scale using a bilinear transform [Shikano 86]. A frame parameter characterizing waveform power (energy per frame), is also computed (as a logarithm of the sum of squares of Hamming-windowed waveform samples). Since "raw" power may vary widely from speaker to speaker, it is normalized by subtracting the maximum power value over all frames obtained in the sentence, from each frame's power value in that sentence. (In a real-time acoustic processing system, a gain-control algorithm would be needed for similar normalization of the power parameter.)

This representation is the same as the one used in the SPHINX system [Lee 89], and is very similar to that used by [Shikano 86] and [Rabiner *et al.* 84].

In speech recognition systems using Minimum Distortion Vector Quantizers (VQ) to encode acoustic frames, such as the SPHINX system [Lee 89], additional acoustic parameters are derived from the cepstrum and power parameters which account for their temporal changes. These are differenced cepstrum coefficients and differenced power computed over a 40-msec time shift (+/- 20 msec time shifts from the current frame). We also compute these parameters (and a number of other parameters characterizing dynamics of the cepstrum and power changes). But we do that in the context of creating acoustic features for constructing Maximum Mutual Information (MMI) encoder trees which are the SSI's method of encoding speech temporal units (frames and segments).

In summary, CEP AP's frame is a 10-msec time interval characterized with 13 parameters (computed over a time window of 20 msec, with the 10-msec lookahead): 12 cepstrum coefficients, and power.

3.2.4.1.3. Approximations to Existing Acoustic Processing Schemes

In our study, we had to alter/approximate some of the processing performed in both schemes (as they are implemented in the existing speech recognition systems). In particular, the pre-emphasis after A-to-D was performed according to the CEP AP algorithm (implemented in [Lee 89]), which differs from that of the FBP AP algorithm (implemented in the SSI speech recognizer).

The gain control algorithm of FBP AP is usually applied to the speech signal prior to the A-to-D conversion; in this study, it was applied after 16-kHz sampling and 14-bit quantization of signal waveform.

Finally, normalization of the power parameter in CEP AP assumed availability of an entire spoken sentence, which is an unrealistic assumption. We have not tried using a "realistic" gain control algorithm for the parameter.

It should be noted that gain control of the input speech signal is not necessary for the CEP AP if an additive gain normalization is applied only to the power parameter. This is due to properties of cepstrum coefficients and power. For the FBP AP, however, a multiplicative gain normalization of input is important because the scale of the magnitude of the filterbank outputs are directly proportional to that of the input speech signal magnitude.

3.2.4.2. Method of Evaluation and Comparison of the FBP and CEP AP Schemes

Two acoustic processing schemes and their corresponding speech representations (FBP AP and CEP AP) were evaluated and compared by applying them to a speaker-independent, large vocabulary continuous speech recognitions task. For such an evaluation, we selected

the DARPA Resource Management (RM) task, with the perplexity 60 word-pair grammar [Pallett 89]. For each of the two APs, a speech recognition model was built using the standard "extended training set" of 4358 sentences from 109 speakers for speaker-independent training of model parameters. All the results were evaluated on 300 independent test sentences from 12 speakers (the June 88 test set). The model architecture used for both APs was identical, and is described below in detail.

3.2.4.3. Speech Recognition Model Architecture

Here we briefly describe the system used in our experiments. Figure 3.2.4.1 summarizes the system architecture.

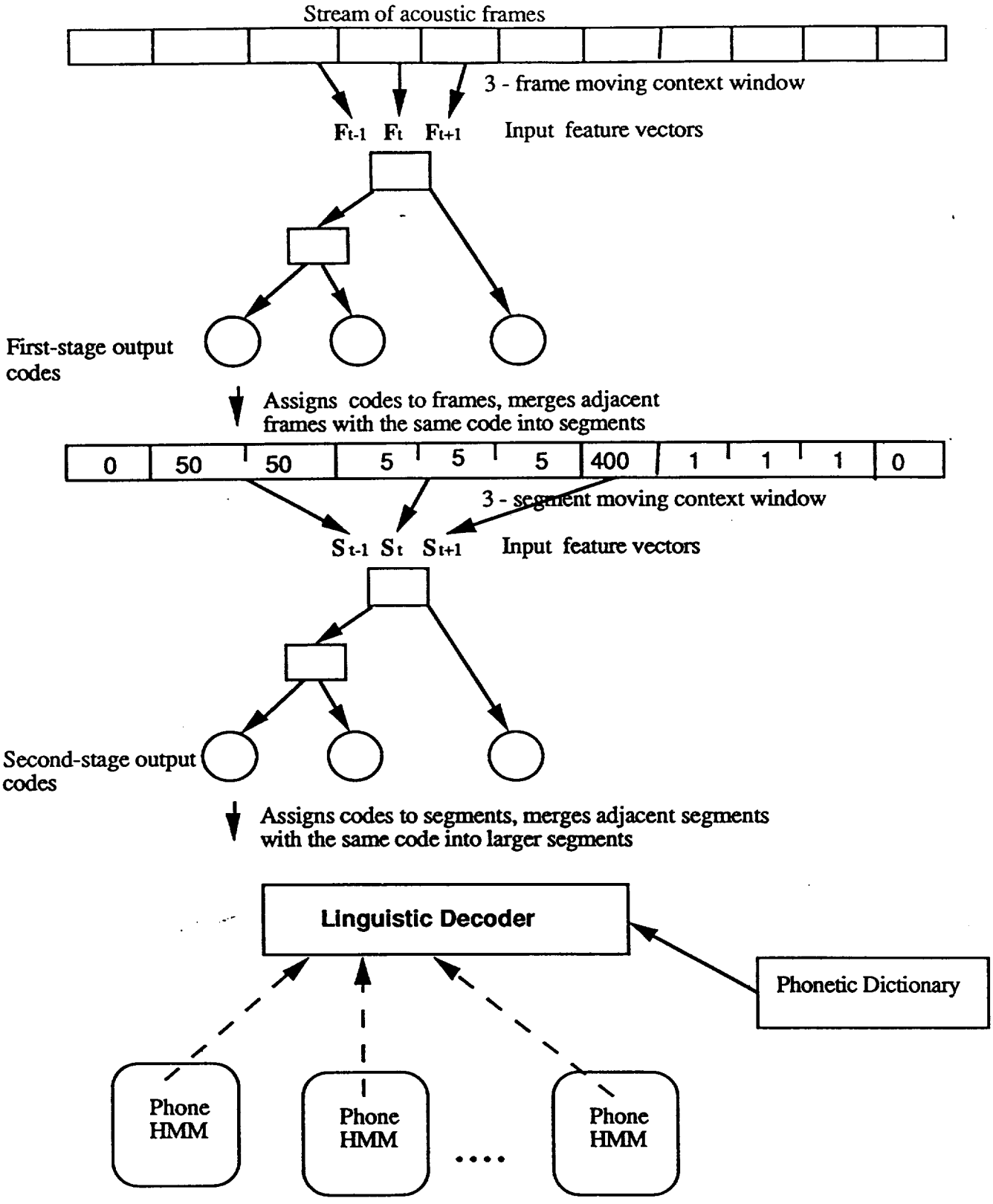


Figure 3.2.4.1: System Architecture (Two-Stage Cascade of MMI Encoder Trees & Phone HHMs)

We train a two-stage cascade of binary-tree structured Maximum Mutual Information (MMI) encoders [Anikst, *et al.* 90]. In the first stage, *frames* are encoded to extract maximum information about their target label classes. Feature vectors used in the tree encoder are frame-based. Continuous runs of frames with the same code are compressed into segments. In the second stage, the resulting *segments* are encoded to extract maximum information about their target label classes (we assign a single target label class per segment). Segment-based acoustic feature vectors are used in the second-stage encoder tree, along with some categorical features based on the phonetic identities discovered by the first-stage encoder tree. Segment duration features are also used. Resulting runs of segments with the same code are again compressed into larger segments.

The stream of segment codes output by the second stage of the cascade is used to train speaker-independent discrete hidden Markov models in a simplified version of the SPHINX system limited to 48 context-independent phonetic HMMs [Lee 89].

3.2.4.3.1. Acoustic Parameterizations

Applying each of the two APs (FBP AP and CEP AP) to the RM training and test speech data, two streams of acoustic frames were created. Each frame has a constant duration of 10 msec, and is characterized by a vector of the acoustic parameters: an FBP AP frame has 23 parameters, and a CEP AP frame, 13 parameters (see Sections 3.2.4.1.1 and 3.2.4.1.2).

3.2.4.3.2. Labelling

Training of the tree-structured MMI encoders is performed using labelled speech data. The set of label classes used for labelling contains 48 classes: there is a unique label class for each of the 48 SPHINX context-independent phones. Labelled frame data for training is obtained via Viterbi alignment using the SPHINX system.

3.2.4.3.3. First-Stage (Frame) MMI Encoder

At the first (frame-coding) stage, frames are encoded in such a way as to convey maximum information about their underlying label class identities. To perform frame encoding, the frame time-sequence is scanned by a "sliding window" covering W frames; in our experiments, we used W = 3.

Several sets of acoustic features derived from the frame acoustic parameters are computed over the frames accessed by the window. These sets include: (a) the acoustic parameters of the center frame of the window; (b) their window-wide (three-frame) averages; (c) first differences of the parameter over 10 msec shift (center frame -left frame), 20 msec shift (right frame -left frame), 40 msec shift (next_to_right frame - prior_to_left frame); and (d) second differences (right frame + left frame -2 * center frame). The frame tree encoder takes as input some (possibly, all) of these features, and outputs a code for the frame at the center of the window. The encoder is trained to maximize the average mutual information between its code alphabet and the alphabet comprised of the 48 target label classes.

The resulting sequence of coded acoustic frames is further processed to form acoustic segments by merging time-contiguous blocks of frames with the same code. Also, the most likely broad phonetic class is assigned to each formed segment, thus introducing an

emergent feature which characterizes a phonetic nature of the code. This feature is included in the input into the encoder tree of the second stage as a "categorical" (nominal) feature.

To obtain a categorical feature based on the phonetic class of a segment, we combined 48 target phonetic classes into nine broad (super)classes, and, for each code, used its most likely broad phonetic class number. The selected set of broad phonetic classes is shown in Table 3.2.4.1 (in the standard notation of the SPHINX phonetic system [Lee 89]).

Class	Phones
0	SIL
1	S SH Z ZH TS JH CH
2	W L
3	V F TH
4	T K P H TD PD KD G B D DH DX DD
5	R ER
6	N M NG
7	AH AE AA AY AO OW OY AW
8	EH EY IH IY AX IX Y UH UW

Table 3.2.4.1: Broad phonetic classes used in a categorical variable for the second-stage tree.

The stream of the acoustic segments with the assigned broad phonetic classes constitutes the input to the segment-coding stage.

3.2.4.3.4. Second-Stage (Segment) MMI Encoder

The second (segment-coding) stage processing is similar to that of the frame-coding stage. Namely, segments are encoded in such a way as to convey maximum information about their underlying phonetic classes.

To perform segment encoding, the stream of segments is scanned by a sliding time window covering three segments (W = 3). A set of pre-defined segment feature vectors is extracted from the acoustic parameters of all the frames encountered in the segments accessed by the window. These segment features include: (a) the averages over each of the frame acoustic parameters computed over the three segments of the window (left, center, right), and those of the 40 msec differenced parameters; (b) the most-likely broad phonetic classes assigned after the first stage to each of the three segments in the window as categorical variables (these variables provide phonetic features complementing the acoustic features); and (c) segment duration features.

The segment encoder tree takes as input these sets of features and outputs a code for the segment in the center of the window. The encoder is trained to maximize the average mutual information between its code alphabet and the alphabet comprised of the 48 target label classes. The target labels for segments are derived from the labels of the constituent frames.

The resulting sequence of coded segments is further processed to form larger segments as in the first stage. The stream of the enlarged segments with the assigned codes constitutes the output of the second stage.

3.2.4.3.5. MMI Training

Our MMI encoders are binary decision trees built to maximize the average mutual information between the phonetic targets and the codes assigned to them. The decision (interior) nodes of the encoder tree are allowed to use linear combinations of feature vectors, as well as unordered categorical features. The task of training such encoders has been extensively addressed in the theory of binary decision trees [Meisel & Michalopoulos 73; Payne & Meisel 77; Breiman *et al.* 84].

The first- and second-stage MMI encoders are trained using labelled data (supervised training). The encoders are trained as binary decision trees using maximization of the average mutual information I(*classes, codes*) between the set of target label *classes* and the set of leaf-node numbers (*codes*), as the training criterion:

I(*classes, codes*) =

$$\sum_{class} \sum_{code} \Pr(class,code) * \log(\Pr(class,code)/(\Pr(class)*\Pr(code))),$$

where Pr(*class,code*) is the joint probability of the *class* and the *code* assigned to a training sample, Pr(*class*) and Pr(*code*) are the marginal *class* and *code* probabilities, respectively.

Training is performed top-down, starting from the root of the binary decision tree. The decision function associated with each decision node of the tree effects the split of the feature space with a hyperplane (for the continuous-valued feature vectors) or a dichotomy of a discrete set (for a categorical feature). The training samples at each node were those which reached that node after passing through predecessor nodes.

Training of the decision function at a node uses as an optimization criterion the reduction in the node's average class entropy. To find a decision hyperplane, we use conjugate gradient based search [Press *et al.* 86] where the gradient of the criterion function with respect to the hyperplane coefficients is computed by replacing the "hard limiter" decision function with a piecewise linear one (the threshold-logic type) and gradually annealing that non-linearity to the hard limiter. Once the optimal coefficients are estimated, we use the hard limiter decision function to send the patterns to the left or the right child node. We don't split a node if the highest reduction in the class entropy attained by the "node split" is less than a certain fraction of the node's class entropy; a final node is a *terminal* node.

After the entire binary tree is created, its performance criterion (i.e. the average mutual information between the set of target classes and the set of the terminal nodes) is evaluated with a combination of the training and independent sets of labelled data. Some nodes are then removed, starting with the current terminal nodes, i.e., the tree is "pruned," to produce a more robust subtree with more accurate estimates of the node-class probabilities. The resulting terminal node numbers are used as codes.

The above training and pruning of the trees was performed utilizing SSI's tree-growing software.

3.2.4.4. MMI Encoders: Training Results

Speech recognition models (the model architecture is shown in Figure 3.2.4.1) were trained for both the FBP AP and CEP AP schemes. Table 3.2.4.2 shows estimates of the average mutual code-phonetic class information for the encoder trees from two stages, along with their other characteristics.

	#codes	#phonetic classes	avg. class entropy, bits	avg. code-class info, bits	info as % class entropy
FBP AP					
Stage 1 Encoder	500	48	5.15	2.08	40.47
Stage 2 Encoder	615	48	5.22	2.30	44.05
CEP AP					
Stage 1 Encoder	547	48	5.20	2.12	40.85
Stage 2 Encoder	636	48	5.21	2.30	44.04

Table 3.2.4.2: Training of Stage Encoder Trees for FBP and CEP APs

Table 3.2.4.3 displays the average number of temporal units (frames or segments) per target phonetic class in the *Per Target* column. The number of segments decreases with each stage of successive temporal compression. In the final segmentation, the number of temporal units per target phonetic class is reduced by a factor of 2 for both models. Temporal compression may lead to the loss of some target phone boundaries (if two or more target phones are merged into one segment). To measure *deletions* of phone boundaries, we compute the percentage of the target phone boundaries lost due to segmentation errors (the *Deleted Bounds* column). As seen from the results in the Table 3.2.4.3, the percentage of the lost target phone boundaries is insignificant (not exceeding 0.2%).

	Segmentation	Per Target	Deleted Bounds
FBP AP			
	Before 1-st stage	8.8 frames	0.00%
	After 1-st stage	5.9 segments	0.05%
	After 2-nd stage	3.3 segments	0.20%
CEP AP			
	Before 1-st stage	8.8 frames	0.00%
	After 1-st stage	5.8 segments	0.05%
	After 2-nd stage	3.3 segments	0.18%

Table 3.2.4.3: Effect of Segmentation

3.2.4.5. Phone Models

The stream of segment codes output by the last (second) stage was used to train (and decode with) the phone HMMs.

The set of phone HMMs used in the study, are 48 context-independent phone HMMs of the phonetic alphabet used in the SPHINX system. This alphabet allows a single-pronunciation-per-word phonetic dictionary. The SPHINX phone HMMs have 7 states, 12 transitions, and 3 distinct pdfs [Lee 89].

Training of the 48 context-independent phone HMMs was performed using the Baum-Welch algorithm (15 iterations, initialized with uniform transition probabilities and output code pdfs.). This was followed by deleted interpolation (the SPHINX training software was used). The RM training data set of 4358 sentences from 109 speakers was used for

speaker-independent HMM training. A small probability floor value was applied, to smooth under-trained probability components.

3.2.4.6. Recognition Test Results

Recognition tests were performed using a Viterbi beam-search decoding algorithm. All results were evaluated on 300 independent test sentences from 12 speakers (the June 88 test set) using the DARPA Resource Management (RM) task's word-pair grammar with perplexity 60 [Pallett 89]. The obtained results are given in Table 3.2.4.4 below. Percent of word accuracy (%word accuracy) is measured as percent of correctly recognized words (%word correct) minus percent words inserted (%word insert.). Average CPU time spent in decoding a sentence is measured for a SUN SPARCstation 1.

AP Scheme	% sent. accuracy	% word accuracy	% word correct	% word subst.	% word delet.	% word insert	CPU sec/sent
FBP AP	26.7	74.6	76.8	17.7	5.5	2.2	22.5
CEP AP	23.3	72.3	74.5	18.8	6.7	2.2	22.7

Table 3.2.4.4: RM Test Recognition Results for FBP and CEP AP Schemes

3.2.4.7. Discussion of the Results and Conclusion

The obtained results (Table 3.2.4.4) indicate that performance of the speech recognition model using FBP AP frame parameters is slightly better than that of the model using CEP AP frame parameters. Examination of the encoder trees performance on the training data for the two models (Table 3.2.4.2) shows that, at each stage, the amounts of the average code-phone mutual information extracted by the encoders of the two models are comparable. Numbers of codes used at each stage by the models' encoders are also close. Finally, the degrees of time compression achieved after each stage, are very close for both models (see Table 3.2.4.3)

Based of the above results, we conclude that the two schemes of frame parameterization are comparable. In view of this conclusion, we will continue with the current FBP AP scheme.

3.2.4.8. Future Work

One advantage of the cepstrum representation that was not exploited in the study is the reduction in dimensionality of the frame acoustic parameter vector achieved by switching from FBP AP (23 parameters) to CEP AP (13 parameters). High dimensionality restricts choices of the acoustic features which can be used, especially in the MMI encoder trees.

In future research, we plan to explore the lower dimensionality of CEP AP, along with looking into ways of reducing the dimensionality of FBP AP without losing important acoustic information.

4. CONSOLIDATION: DELIVERY AND INSTALLATION

4.1. OVERVIEW

At the end of the contract, a consolidated system was put together, delivered and installed at NASA Johnson Space Center. This system includes results from many of the studies performed under this contract. In this section we give a summary of what was included in the final delivered system and some notes about how the installation was received.

4.2. SUMMARY OF FINAL DELIVERED SYSTEM

We summarize here the changes that have been put into the delivered system that are a direct result of our research and development on this contract. These are grouped into five areas: acoustic processing, phonetic recognition, phonetic decoding, model building, and model adaptation.

Acoustic Processing:

* constant frame rate acoustic processing
* acoustic parameter smoothing
* acoustic parameterization via filterbank scheme (retained)
* headset input device

Phonetic Recognition:

* generic speaker model improvements, particularly release 3.3 phonetic system improvements (yielding major accuracy improvements)

Phonetic Decoding:

* decoder reengineering for parallel processing modifications (yielding major decoding time reductions)
* theoretical results predicting performance of a parallel implementation

Model Building:

* 8 custom profiled models for NASA speakers
* 2 generic models (male and female) for Southern dialect, as represented by NASA speakers (dictionary changes unnecessary)

Model Adaptation:

* delivered profiler for automatic model adaptation
* profiler scheme suitable for improving noise robustness (see section 3.2.1)

4.3. INSTALLATION NOTES

The system installed at NASA JSC was SSI's general release 3.3.1. The speed and accuracy improvements of this release were quite noticeable, confirming the predictions of system improvements in these areas.

Included in this release is the first real release of the profiler. Our tests for noise sensitivity (section 3.2.1) showed that using the profiler can improve the noise robustness of the profiled models, if the speech data for profiling is collected in a similar environment to the target application. The final models delivered to NASA did not include any of this profiling for noise robustness. However, with the delivery of the profiler, this type of adaptation is available for any application.

To improve the noise robustness, this is the method we would recommend for any specific new application(s) that NASA wants to implement: Use the profiler on speech data for that application in that application's environment; use one of the NASA generic models as the bootstrap model; and produce new model(s) customized for that application, environment, and/or speaker(s). This method will improve the noise robustness for the type of noise actually encountered, as well as adapt the model(s) to the specific speakers and/or applications.

4.4. CONCLUSIONS

We have consolidated the results of our research together into a final system. This system was delivered and installed at NASA JSC, demonstrating our ability to translate research results into delivered improvements. A version of the profiler was also delivered, giving the benefits of model adaptation technology to the application developer.

5. REFERENCES

M.T. Anikst, W.S. Meisel, M.C. Soares and K-F. Lee, "Experiments with Tree-Structured MMI Encoders on the RM Task," *1990 DARPA Workshop on Speech and Natural Language*, June 1990.

L.R. Bahl, F. Jelinek and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," in *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, pp. 179-190, March 1983.

R. Bisiani, T. Anantharaman and L. Butcher, "BEAM: An Accelerator for Speech Recognition," Report CMU-CS-89-102, Computer Science Department, Carnegie Mellon University, Pittsburgh, Penn., January, 1989.

G. Borden and K. Harris, *Speech Science Primer: Physiology, Acoustics, and Perception of Speech*, 2nd Ed., Baltimore, Williams and Wilkins, 1984.

L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth International Group, Belmont, Calif., 1984.

F. Charot, P. Frison, and P. Quinton, "Systolic Architectures for Connected Speech Recognition," in *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 765-779, August 1986.

S.B. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously-Spoken Sentences", *IEEE Trans. Acoust. Speech Sig. Process. ASSP-28*, p. 354 , 1980.

Digital Equipment Corporation, *Guide to Multiprocessing on VAX/VMS*, Maynard, Mass., April 1986a.

Digital Equipment Corporation, *Introduction to Parallel Programming*, Maynard, Mass., September 1986b.

C. Godin and P. Lockwood, "DTW Schemes for Continuous Speech Recognition: A Unified View," in *Computer Speech and Language*, (1989) 3, pp. 196-198.

R. Hudson, *Sociolinguistics*, Cambridge, Cambridge Univ. Press, 1980.

S.W. Kumar, *High-Speed Computing in AI: A Technology Assessment*, The Boeing Company, 1986.

P. Ladefoged, *Elements of Acoustic Phonetics*, Chicago, Univ. of Chicago Press, 1971.

P. Ladefoged, *A Course in Phonetics*, Harcourt Brace Jovanovich, Inc., New York, New York, 1982.

C.H. Lee, L.R. Rabiner, R. Pieraccini, and J.G. Wilpon, "Acoustic Modeling for Large Vocabulary Speech Recognition", in *Computer Speech and Language* 4, pp. 127-165, 1990.

K-F. Lee, *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System*, Ph.D. thesis, Computer Science Department, Carnegie Mellon University, April, 1988.

K-F. Lee, *Automatic Speech Recognition: The Development of the SPHINX System.* Kluwer Academic Publishers, Boston, 1989.

B. Lowerre and R. Reddy, "The Harpy Speech Understanding System," in W.A. Lea, ed., *Trends in Speech Recognition*, pp. 15.1-15.9, Academic Press, 1980.

J.D. Markel and A.H. Gray., *Linear Prediction of Speech*, Springer-Verlag, Berlin, 1976.

W.S. Meisel and D.A. Michalopoulos, "A Partitioning Algorithm with Application in Pattern Classification, Piecewise-Constant Approximation, and the Optimization of Decision Trees," *IEEE Trans. on Computers*, January 1973.

W.S. Meisel, P.C. Shinn, D.J. Trawick and W.A. Wittenstein, "Speech Representation and Speech Understanding," DARPA Phase I SBIR Final Report, Reference DAAH01-87-C-0953, March 1988.

W.S. Meisel, M.P. Fortunato and W.D. Michalek, "A Phonetically-Based Speech Recognition System," *Speech Technology*, pp. 44-48, Apr/May 1989.

D.S. Pallett, "Benchmark Tests for DARPA Performance Evaluations," *Proc. ICASSP 89*, pp. 536-539, May 1989.

D.B. Paul, "The Lincoln Robust Continuous Speech Recognizer", *Proc. ICASSP 89*, May 1989.

H.J. Payne and W.S. Meisel, "An Algorithm for Constructing Optimal Binary Decision Trees," *IEEE Trans. on Computers*, September 1977.

W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.

L.R. Rabiner, K.C. Pan and F.K. Soong, "On Performance of Isolated Word Speech Recognizers Using Vector Quantization and Temporal Energy Contours," *AT&T Bell Lab. Tech. J., vol. 63, no. 7, pp. 1245-1260*, Sept. 1984.

M.J. Russell, K.M. Ponting, S.M. Peeling, S.R. Browning, J.S. Bridle, and R.K. Moore, "The ARM Continuous Speech Recognition System", *Proc. ICASSP 90*, pp. 69-72, April 1990.

K. Shikano, "Evaluation of LPC Spectral Matching Measures for Phonetic Unit Recognition," *Tech. Rep., Comp. Science Dept., Carnegie Mellon Univ.*, May 1986.

G. Stainhaouer and G. Carayannis, "New Parallel Implementations for DTW Algorithms," in *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-38, pp. 705-711, April 1990.

S.J. Stolfo, Z. Galil, K. McKeown and R. Mills, "Speech Recognition in Parallel," in *Proceedings of the DARPA Speech and Natural Language Workshop*, Harwich Port, Mass., October 1989.

Sun Microsystems, Inc., *Network Programming*, Part Number: 800-1779-10, Revision A, May 1988.

E. Zwicker, "Subdivision of the Audible Frequency Range into Critical Bands," *J. Acoust. Soc. of America, 33:248*, Feb. 1961.